VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Testování webové aplikace se zaměřením na porovnání kvality verzí při použití manuálních a automatizovaných testů

Testing of web application with a focus on quality comparison of versions using manual and automated tests

2017 Bc. Mário Červeň

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Mário Červeň**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Testování webové aplikace se zaměřením na porovnání kvality verzí při použití manuálních a automatizovaných testů
Testing of Web Application with a Focus on Quality Comparison of Versions Using Manual and Automated Tests

Jazyk vypracování: čeština

Zásady pro vypracování:

Diplomová práce je zaměřená na testování webové aplikace ve společnosti NetDirect s.r.o. Cílem práce bude vytvoření a porovnání kvality testované aplikace při použití manuálních a automatizovaných testů.

1. Představení společnosti NetDirect s.r.o., e-shopové platformy FastCentrik a popis aktuálního procesu vývoje ve společnosti.
2. Obecně o testování SW.
3. Obecné porovnání manuálního a automatizovaného testování.
4. Přehled vhodných testovacích nástrojů, jejich popis, porovnání funkcí a cen, vhodnost použití ve vývojovém oddělení společnosti.
5. Popis testované webové aplikace.
6. Seznam testovacích scénářů, popisy, rozdělení (minimálně 20).
7. Automatizované testy pro jednotlivé testovací scénáře (minimálně 20).
8. Doporučení pro psaní programového kódu v závislosti na použití testovacího nástroje.
9. Zapojení automatizovaných testů do procesu Buildu.
10. Statistika neodhalených chyb při použití automatizovaných testů nad historickými verzemi webové aplikace.
11. Porovnání kvality aplikace při použití manuálních a automatizovaných testů.
12. Vyhodnocení výsledků porovnání.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Rogozný**

Konzultant diplomové práce: Ing. David Ježek, Ph.D.

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

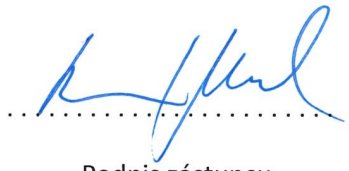
V Ostrave dňa 11. 4. 2017 .....

Podpis študenta

Prehlásenie zástupcu spolupracujúcej právnickej alebo fyzickej osoby

Súhlasím so zverejnením tejto diplomovej práce podľa požiadaviek čl. 26, odst. 9 Študijného a skúškového rádu pre štúdium v magisterských programoch VŠB-TU Ostrava.

V Ostrave dňa 11. 4. 2017



Podpis zástupcu

Touto cestou chcem poďakovať Ing. Davidovi Ježkovi, Ph.D. za ochotu, odborné rady a ústretový prístup na konzultáciách. Ďalej by som chcel poďakovať Ing. Michalovi Rogoznému za poskytnuté informácie a za umožnenie vzniku tejto práce.

Abstrakt

Táto diplomová práca je zameraná na testovanie webovej aplikácie a na porovnávanie kvality verzií aplikácie pri použití automatizovaných a manuálnych testov. Súčasťou praktickej časti bola práca vo firme zameraná na testovanie vysoko využívanej webovej aplikácie. V tejto diplomovej práci sa nachádza teória testovania, definície používaných pojmov testovania, popis použitých nástrojov a testovanej webovej aplikácie. Ďalej som popísal rozdelenie vytvorených testovacích scenárov. Popísal som aj procesy automatizácie testu v Test Studiu, tvorby testovacích zoznamov, integrovanie automatizovaných testov do procesu zostavenia a proces testovania verzií aplikácie. Hlavným cieľom bolo vytvorenie automatizovaných testov v Test Studiu pre dôležité časti aplikácie. Ďalším cieľom bolo pomocou automatizovaných a manuálnych testov otestovať verzie aplikácie a porovnať ich kvalitu.

Kľúčové slová: testovanie softvéru, kvalita softvéru, automatizované testovanie, manuálne testovanie, Telerik Test Studio, C#, TFS

Abstract

This master's thesis is focused on web application testing and quality comparison of application versions using automated and manual tests. The practical part was focused on testing of a web application. I described theory, terms and tools used for testing and web application I used for testing. Next I described the distribution of the test scenarios I created. I also described the processes of automation in Test Studio, creating test lists, integrating automated tests into the build process and the process of testing application versions. The main objective was to create automated tests in Test Studio for the important parts of the application. Another objective was using automated and manual tests to test application versions and to compare their quality.

Keywords: software testing, software quality, automatized testing, manual testing, Telerik Test Studio, C#, TFS

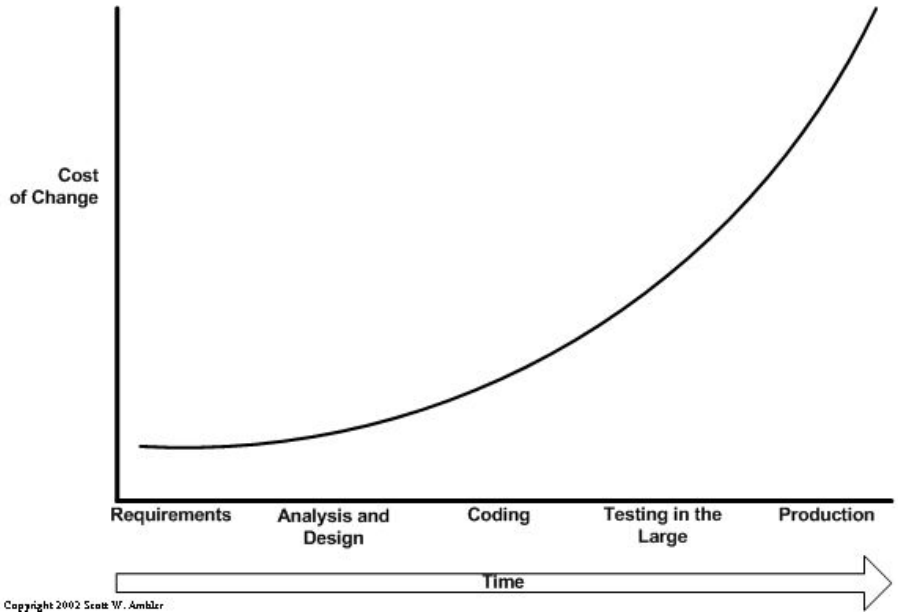
Zoznam použitých skratiek a symbolov

AJAX	–	Asynchronous JavaScript and XML – metóda asynchronného spracovania webstránok
DOM	–	Document Object Model – objektovo orientovaná reprezentácia XML, HTML alebo XHTML dokumentu
GUI	–	Graphic User Interface – grafické užívateľské rozhranie
TFS	–	Team Foundation Server
UI	–	User Interface – užívateľské rozhranie
WPF	–	Windows Presentation Foundation – knižnica tried pre tvorbu grafického užívateľského rozhrania pre Windows aplikácie
WYSIWYG	–	What You See Is What You Get – princíp prenosu informácie, v ktorom dostaneme rovnaký výstup ako ten, ktorý vidíme na vstupe

Obsah

1 Úvod

Jedným z najväčších problémov softvéru je jeho chybovosť. Vyvíjať softvér bez chýb je prakticky nemožné. Vždy preto musíme predpokladať, že náš softvér má chyby. Okrem nášho zdrojového kódu sa chyby môžu vyskytovať aj v knižniciach, ktoré používame a môžu byť aj v operačnom systéme, na ktorom je softvér spúšťaný. V prípade webových aplikácií sa často stáva, že webstránka na niektorých prehliadačoch nefunguje správne. Okrem toho sú známe chyby v prekladačoch, že sa zle preloží časť zdrojového kódu. Pri niektorých zdrojoch chýb nemáme prístup ku zdrojovým kódom, a preto ich nedokážeme všetky opraviť. Hľadanie spomínaných chýb je jednou z aktivít testovania softvéru.



Obrázok 1: Cena opravy chyby v čase [2]

Z obrázku 1 vidíme, že problémy je exponenciálne drahšie riešiť po nasadení ako pred ním. Je teda v záujme všetkých zúčastnených strán pri tvorbe softvéru testovať čo najskôr a často. Znížením času medzi vývojom a overovaním správnosti softvéru znížime náklady na opravy chýb a v konečnom dôsledku zvýšime náš zisk. Testovanie je preto neoddeliteľnou súčasťou vývoja softvéru a taktiež jedným z aspektov maximalizácie zisku.

1.1 Zameranie, cieľ a štruktúra práce

Táto práca je zameraná na testovanie webovej aplikácie a na porovnávanie kvality verzií aplikácie pri použití automatizovaných a manuálnych testov. Kvalitu som overoval za použitia manuálnych a automatizovaných testov. Tvoril som testy pre funkčné (testovanie funkcionality) testovanie UI webovej aplikácie. Okrem toho som sa zameral na tes-

tovací nástroj Test Studio a prácu v ňom. Testovanou webovou aplikáciou je platforma slúžiaca pre tvorbu internetových obchodov. Táto platforma je vysoko využívaná, aktuálne na nej funguje viac než 1300 internetových obchodov.

Hlavným cieľom bolo vytvorenie automatizovaných testov v Test Studiu pre dôležité časti aplikácie. Ďalším cieľom bolo pomocou vytvorených automatizovaných a manuálnych testov otestovať verzie aplikácie a porovnať ich kvalitu.

Kvôli rozsahu aplikácie samozrejme nebolo možné zabezpečiť testami jej plné pokrytie, a preto som sa zameral na testovanie len tých najdôležitejčích častí aplikácie. Vytvorené testovacie scenáre som preto rozdelil do niekoľkých skupín. Rozdelenie testovacích scenárov a testovanú aplikáciu bližšie popisujem v tejto práci. V osobitnej kapitole potom vysvetľujem potrebnú teóriu a pojmy, s ktorými sa pri testovaní stretávame.

Automatizované testy som tvoril v Test Studiu. Tieto testy som rozdelil do niekoľkých testovacích zoznamov, ktoré sa dajú spúšťať osobitne. Automatizované testy som potom integroval do procesu zostavenia. Nakoniec som pomocou automatizovaných aj manuálnych testov testoval 5 rôznych verzií webovej aplikácie. Výsledky som zapísal a kvalitu jednotlivých verzií porovnal.

Táto práca je rozdelená do nasledujúcich kapitol:

- **NetDirect s.r.o** – táto kapitola hovorí o firme, pre ktorú som pracoval.
- **Testovanie softvéru** – je tu vysvetlená teória o testovaní softvéru, jeho delenie a dôležité pojmy.
- **Nástroje pre automatizované testovanie** – v kapitole sú popísané a porovnané jedny z najpoužívanejších testovacích nástrojov.
- **Použité nástroje** – sa zaoberá jednotlivými nástrojmi, ktoré som pri práci používal.
- **Inštalácia a konfigurácia použitého softvéru** – v kapitole je postup inštalácie a konfigurácie softvéru, ktorý som používal.
- **Testovaná webová aplikácia** – je uvedená webová aplikácia, ktorú som testoval.
- **Vytvorené testovacie scenáre** – hovorí o vytvorených testovacích scenároch a ich rozdelení.
- **Automatizované testovanie v Test Studiu** – popisuje moju tvorbu automatizovaných testov v Test Studiu a ďalšie činnosti s tým spojené.
- **Testovacie zoznamy** – uvádza testovacie zoznamy a postup ich tvorby.
- **Odporúčania pre písanie zdrojového kódu vývojármi** – obsahuje najčastejšie chyby vývojárov a odporúčané riešenia týchto chýb. Spomínam tu aj ako tieto chyby ovplyvňujú vybrané testovacie nástroje.
- **Integrovanie automatizovaných testov do procesu zostavenia** – kapitola, v ktorej opisujem postup integrácie automatizovaných testov do procesu zostavenia.

- **Testovanie verzií webovej aplikácie** – obsahuje opis priebehu testovania verzií, výsledky testovania, popis úprav testov a prínos testovania verzií.
- **Záver** – obsahuje zhodnotenie, výsledky a prínos vykonanej práce. Okrem toho sú tu uvedené znalosti a skúsenosti, ktoré som touto prácou nadobudol.

2 NetDirect s.r.o.

Časťou mojej diplomovej práce je činnosť, ktorú som vykonával vo firme NetDirect s.r.o [6]. Je to spoločnosť sídliaca v Ostrave a v Prahe s dlhoročnými skúsenosťami s tvorbou a prenájmom internetových obchodov a web stránok rôzneho druhu. Dôležitá ja pre nich optimalizácia a stabilita web stránok a zabezpečenie ich rýchleho chodu aj pri vysokom zaťažení. Pri tvorbe webových aplikácií využívajú najnovšie technológie, ktoré sa snažia udržovať vždy aktuálne. Pre svojich zákazníkov ponúkajú aj ďalšie služby, ako poradenstvo marketingu, v optimalizácii pre vyhľadávače a PPC (Pay Per Click), čo je jedným zo spôsobov platenia za reklamu na internete.

Vďaka využívaniu najnovších technológií, zamestnávaniu certifikovaných odborníkov a odbornej podpory si vyslúžili od spoločnosti Microsoft prestížny titul Microsoft Gold Certified Partner. Takto certifikované spoločnosti sú označované za „vysoko akreditovaných a nezávislých poskytovateľov technickej podpory spoločnosti Microsoft“. Firma si dlhodobo udržuje aj certifikát ISO 9001. Tento certifikát je medzinárodnou normou týkajúcou sa systému riadenia kvality a pokrýva všetky oblasti podnikania, od výrobných činnosti až po poskytovanie služieb. Okrem toho je firma členom IT clusteru a Asociácie pre elektronickú komerciu.

Hlavnými produktami firmy sú:

- FastCentrik – platforma pre internetové obchody.
- ShopCentrik – internetové obchody na mieru.
- MediaCentrik – tvorba webstránok, portálov a rôznych systémov pre správu obsahu.

2.1 Vývojový proces

Vo firme je zaužívaný agilný vývoj formou Scrumu. **Scrum** [3] je inkrementálnym a iteratívnym rámcom procesu pre projektový manažment. Jeho cieľom je dodávať pre zákazníkov najdôležitejšie prvky čo najskôr. Organizácie si ho zvyčajne prispôsobujú svojim vlastným preferenciám. Tak je to aj v našej organizácii. Pre riadenie scrumu využívame TFS.

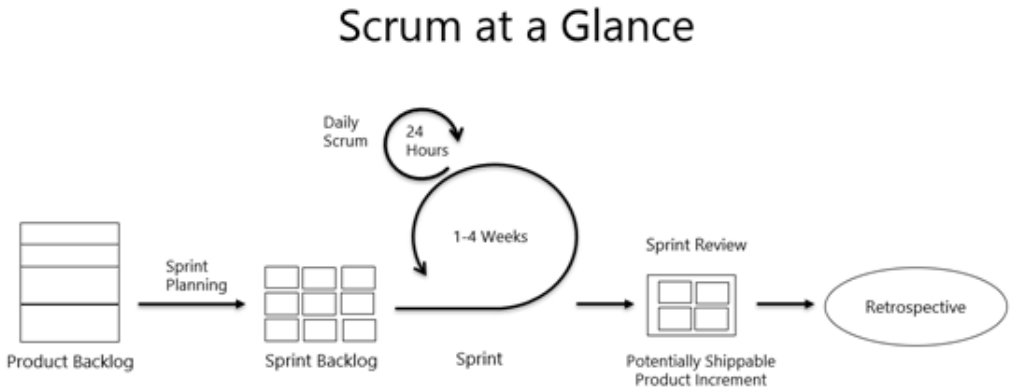
Požiadavky sú zhromažďované, spravované a prioritizované v zozname nazývanom produktový backlog. Ten je typicky spravovaný vlastníkom produktu.

Počas plánovania sprintu [4] vlastník produktu vyberie časť požiadaviek z produktového backlogu do backlogu sprintu. Ak už je backlog sprioritizovaný, postupuje sa zhora. Vlastník produktu s tímom diskutuje náročnosť jednotlivých úloh a naplánuje sa ich implementácia.

Tím sa počas iterácie (tzv. sprintu) stretáva každý deň a zdieľa informácie o stave úloh, plánoch na aktuálny deň a problémoch, ktoré sa vyskytli. Tieto problémy eviduje tzv. Scrum Master. Ten zároveň hľadá riešenia a organizuje proces Scrumu. Sprint typicky trvá 1 až 4 týždne. V našom tíme sú to 2 týždne.

Počas sprintu navyše prebiehajú aj pravidelné stretnutia, ktorým sa hovorí grooming. Ich cieľom je zaistiť, aby tím porozumel celému backlogu. Tím sa na týchto stretnutiach oboznamuje so zmenami, určujú sa možné riziká a odhaduje sa náročnosť implementácie. Väčšinou na nich riešime technické aspekty nejakého väčšieho problému alebo úlohy.

Na konci sprintu by mal byť produkt pripravený na odovzdanie zákazníkovi. Výsledné zmeny produktu sú v našom prípade vydané ako tzv. balíček. Nasleduje retrospektíva, čo je stretnutie, na ktorom sa zrecenzuje čo počas sprintu fungovalo dobre, čo by sa mohlo zlepšiť a čo pre toto zlepšenie môžu jednotlivci tímu urobiť.



Obrázok 2: Vizualizácia Scrumu [5]

3 Testovanie softvéru

Testovanie softvéru je proces, pri ktorom vyhodnocujeme celý systém alebo niektoré jeho časti s cieľom zistiť, či sú splnené požiadavky, ktoré boli pred tým špecifikované. Zisťujeme pomocou neho kvalitu softvéru. Kvalita softvéru je miera dodržania špecifikovaných požiadaviek, potrieb a očakávaní. Má niekoľko dimenzií, ktoré môžme merať. Testovanie kvality softvéru sa vykonáva najmä pre tieto dimenzie kvality [9]:

- **Funkcionalita** – je schopnosť softvéru vykonávať špecifikované alebo požadované funkcie.
- **Použiteľnosť** – je stupeň jednoduchosti použitia softvéru.
- **Spoľahlivosť** – je schopnosť softvéru vykonávať požadované funkcie za daných podmienok po uvedenú dobu bez akýchkoľvek chýb.
- **Výkon** – je rýchlosť reakcie softvéru pod určitou záťažou.

Testovanie softvéru je dôležité z nasledujúcich dôvodov [8]:

- Je potrebné k nájdeniu vád, ktoré vznikli počas samotného vývoja.
- Pomáha zaručiť spokojnosť používateľov produktu.
- Je nevyhnutné k zaručeniu kvality softvéru.
- Zvýšením kvality softvéru znižuje náklady na jeho údržbu.
- Umožňuje odhalenie nedostatkov vo výkone, a pomáha tak k zvýšeniu výkonu softvéru.
- Je potrebné k zaručeniu spoľahlivosti pre rôzne prostredia.
- Pomáha znižovať počet zlyhaní softvéru.

3.1 Základné pojmy

Aby sme si pri testovaní softvéru navzájom rozumeli, je potreba si najskôr definovať základné pojmy [7].

3.1.1 Chyba

Chyba (angl. error) je ľudská činnosť, ktorej výsledkom je vada.

3.1.2 Vada

Vada (angl. fault, bug) je softvérový nedostatok, ktorý môže spôsobiť zlyhanie.

3.1.3 Zlyhanie

Zlyhanie (angl. failure) je neočakávané správanie softvéru.

3.1.4 Ladenie

Ladenie (angl. debugging) je hľadanie a odstraňovanie vád softvéru.

3.1.5 Zaistenie kvality

Zaistenie kvality [12] sú aktivity zameriavajúce sa na zabezpečenie kvality procesov, ktorými je produkt vyvíjaný. Zaistenie kvality sa dosahuje prevenciou vád. Zodpovednosť za zaistenie kvality majú teda na starosti osoby podieľajúce sa na vývoji produktu. Zaistenie kvality sa orientuje na proces. Zlepšuje proces vývoja a testovania s cieľom, aby sa počas vývoja nezvyšoval počet vád.

Ako príklad zaistenia kvality sa dá uviesť **verifikácia**. Verifikácia je proces vyhodnocovania, či je produkt vyvíjaný podľa formulovaných požiadaviek.

3.1.6 Riadenie kvality

Riadenie kvality je proces zameriavajúci sa na určenie vád a na výsledný produkt. Riadenie kvality sa na rozdiel od zaistenia kvality orientuje na produkt. Vykonáva sa inšpekciou a overovaním produktu s cieľom určiť jeho vady. Riadenie kvality majú zvyčajne na starosti osoby testujúce produkt.

Príkladom riadenia kvality je **validácia**. Validácia je vyhodnocovanie, či produkt spĺňa požiadavky zákazníka.

3.2 Delenie testovania softvéru

Existuje niekoľko typov delenia testovania. Základné delenie rozdeľuje testovanie podľa testovanej dimenzie kvality, a to na funkčné a nefunkčné testovanie.

3.2.1 Funkčné testovanie

Funkčné testovanie je zamerané na testovanie funkcionality softvéru. Overuje zhodu funkcií softvéru s požiadavkami a kontroluje fungovanie softvéru na rôznych softvérových a hardvérových platformách.

3.2.2 Nefunkčné testovanie

Nefunkčné testovanie je testovanie nefunkčných požiadaviek. Patria tu všetky dimenzie kvality okrem funkcionality. Nefunkčné požiadavky určujú vlastnosti a obmedzenia softvéru. Testovať tak môžeme použiteľnosť, bezpečnosť, dostupnosť, kompatibilitu, preveditelnosť inštalácie, výkonnosť a iné. Výkonnosť je z nefunkčných požiadaviek testovaná najčastejšie, a preto si povieme viac o výkonnostnom testovaní.

Výkonnostné testovanie testuje nefunkčné požiadavky softvéru, ako sú doba odozvy a priepustnosť pod istým zaťažením. Radí sa tak medzi typy nefunkčného testovania. Medzi výkonnostné testovanie patrí záťažové a stresové testovanie.

Záťažové testovanie je technika testovania, pomocou ktorej skúmame správanie systému a meriame jeho odozvu za bežného až najvyššieho očakávaného zaťaženia. Záťažovým testovaním chceme ukázať, že systém zvládne požadovanú záťaž.

Stresové testovanie je technikou, pri ktorej vystavujeme systém stresu s cieľom zistiť bod záťaže, pri ktorom dôjde k zlyhaniu. Tento bod zistíme preťažovaním systému až do jeho zlyhania. Pomocou stresového testovania určujeme softvérovú robustnosť.

Pri stresovom testovaní je dôležitý pojem stres. Tento pojem vyjadruje vrcholovú záťaž v krátkom čase.

3.3 Úrovne testovania softvéru

Testovanie má 4 úrovne, kde každá nasledujúca úroveň sa zaoberá väčšími časťami systému.

3.3.1 Jednotkové testovanie

U jednotkového testovania sú testované jednotlivé softvérové jednotky alebo komponenty softvéru za účelom overiť, či fungujú podľa požiadaviek.

3.3.2 Integračné testovanie

Integračné testovanie by sa malo vykonávať po jednotkovom testovaní. Otestované softvérové jednotky sa spájajú do komponent, ktoré sa následne otestujú. Otestované komponenty sa ďalej zlučujú do subsystémov, ktoré sa tiež testujú. Cieľom je zistiť prípadný nesúlad medzi integrovanými jednotkami.

3.3.3 Systémové testovanie

Systémové testovanie nasleduje po integračnom testovaní, kedy je systém plne integrovaný. Systémové testovanie je tak vykonávané nad kompletným a integrovaným systémom. Snahou je overiť, či systém spĺňa požiadavky.

3.3.4 Akceptačné testovanie

Akceptačné testovanie je z pravidla vykonávané používateľmi alebo zákazníkmi pre určenie, či systém spĺňa akceptačné kritériá. Vychádzajú zo špecifikácie a vnímajú aplikáciu ako celok.

3.4 Manuálne testovanie

Manuálne testovanie je proces testovania softvéru, kde softvérový testery ručne vykonávajú testovacie prípady za cieľom nájdenia vád. Každá aplikácia musí byť najskôr manuálne otestovaná pred tým, než bude testovanie zautomatizované. Manuálne testovanie je nevyhnutné pre kontrolu uskutočniteľnosti automatizácie. Čo činí manuálne testovanie významným je jedna zo zásad testovania, ktorá vraví, že „stopercentná automatizácia nie je možná“. Oproti automatizovanému testovaniu je manuálne náchylnejšie na chyby kvôli ľudskému faktoru.

3.5 Automatizované testovanie

Automatizované testovanie je proces testovania softvéru, pri ktorom spúšťame testovacie skripty. Tie vznikajú automatizáciou manuálneho testu alebo konkrétnejšie krokov testovacieho prípadu prislúchajúcich danému manuálnemu testu. Okrem toho sa dnes už dá automatizovať takmer celý proces testovania softvéru. K samotnej automatizácii krokov sa používajú softvérové nástroje určené k tejto činnosti. Keďže stopercentná automatizácia nie je možná a sú prípady, kedy je manuálne testovanie preferované, cieľom automatizácie nie je manuálne testovanie úplne eliminovať, ale znížiť počet testovacích prípadov, ktoré sú vykonávané manuálne.

Automatizované testovanie je dôležité najmä pre regresné testovanie, pri ktorom sa znižujú prevádzkové náklady testovania a zvyšuje sa účinnosť testovania. Regresné testovanie vzniklo z dôvodu, že pri vývoji softvéru sa často stáva, že opravením existujúcich vád vzniknú nové. Je to typ testovania softvéru, pri ktorom spúšťame testy s cieľom ubezpečiť sa, že vady, ktoré boli nájdené a opravené sa v softvéri už nevyskytujú a že nevznikli nové vady.

3.6 Porovnanie automatizovaného a manuálneho testovania

Nemôžeme povedať, či je lepšie manuálne alebo automatizované testovanie, pretože každé z nich je vhodné v odlišných situáciách. Povieme si aspoň aké presne tieto situácie sú.

Manuálne testovanie by sme mali vykonávať v nasledujúcich prípadoch:

- Ak testujeme danú funkcionalitu po prvý raz.
- Ak nám stačí daný test spustiť len jedenkrát a v budúcnosti nebude opakovaný.
- Ak vykonávame skúšobný test.
- Ak sa požiadavky často menia.

Automatizované testovanie je vhodné použiť v týchto prípadoch:

- Ak testy potrebujeme spúšťať opakovane.

- Ak chceme testy spúšťať s rôznymi testovacími dátami.
- Ak sú testy časovo náročné alebo manuálne ťažko uskutočniteľné.
- Pre výkonnostné testy. Pri týchto testoch treba simulovať viacero používateľov využívajúcich aplikáciu súčasne.

Väčšina času pri manuálnom testovaní zaberá hľadanie vád. Naopak pri automatizovanom testovaní je tento čas minimálny a navyše testy môžeme nechať spustené aj bez dohľadu. Pri automatizovanom testovaní najviac trvá návrh automatizácie, programovanie testov, ich údržba a úpravy. Z dlhodobého hľadiska je však výhodnejšie. Finančná investícia do automatizovaných testov sa taktiež vypláca z dlhodobého hľadiska, pretože opakované spúšťanie testov nevyžaduje žiadne dodatočné náklady. Okrem toho pri automatizovanom testovaní odpadá ľudský faktor, a tak je z tohto pohľadu spoľahlivejšie.

3.7 Artefakty testovania softvéru

Artefakt je v životnom cykle vývoja softvéru akýkoľvek vedľajší produkt, ktorý je priamo alebo nepriamo vytváraný v procese vývoja softvéru. Môžu to byť napr. dokument, diagram, zdrojový kód, digitálny súbor a pod. Pri testovaní sa najčastejšie stretávame s artefaktmi dokumentácie testovania, zdrojovými kódmi a súbormi testov, vstupnými a výstupnými artefaktmi testovania.

3.7.1 Testovací plán

Testovací plán stanovuje stratégiu, ktorá bude použitá na testovanie aplikácie, prostriedky, ktoré budú použité, testovacie prostredie, v ktorom bude prebiehať testovanie, obmedzenia a časový rozvrh testovacích aktivít.

3.7.2 Testovací scenár

Testovací scenár [10] je výrok popisujúci, ktorú oblasť (zvyčajne funkciu) aplikácie budeme testovať. Testovací scenár a testovací prípad sa často zamieňajú, avšak testovací scenár môže obsahovať jeden alebo viac testovacích prípadov.

3.7.3 Testovací prípad

Testovací prípad [1] podľa ANSI/IEEE 829 „dokumentuje skutočné hodnoty vstupov spolu s očakávanými výstupmi a identifikuje obmedzenia testovacej procedúry“. Hlavným cieľom tejto činnosti je zistiť, či softvér uspeje alebo zlyhá v zmysle funkčnosti a ďalších aspektov. K testovaciemu prípadu býva priradený jeden alebo viac testovacích skriptov. Testovací prípad nemá pevne danú šablónu, avšak väčšinou obsahuje nasledujúce informácie:

- ID – unikátny identifikátor, pomocou ktorého sa môžeme na daný testovací prípad odkazovať,

- testovaný prvok – názov testovaného prvku,
- účel – účel testovaného prvku,
- priorita – priorita testu,
- predpoklady – špecifikujú vstupné hardvérové a softvérové požiadavky, testovacie nástroje, zamestnancov a iné požiadavky k vykonaniu testovacieho prípadu,
- vstupné testovacie dáta – dáta používané pri spúšťaní testu,
- kroky – sekvencia krokov, ktoré budú vykonávané,
- očakávané výsledky – očakávané výsledky po vykonaní testovacieho prípadu.

3.7.4 Testovacie dáta

Testovacie dáta sú dáta, ktoré sa používajú pri testoch na potvrdenie očakávaného výsledku. Tieto dáta môžu byť platné aj neplatné. Neplatné vstupné dáta zvyčajne slúžia na overenie správnosti správania softvéru.

3.7.5 Testovací skript

Testovací skript (nazývaný aj testovacia procedúra) je usporiadaná sada inštrukcií, ktoré sú vykonávané v danom poradí za účelom overenia, že systém sa správa podľa očakávaní. Jeden testovací skript môže byť používaný viacerými testovacími prípadmi.

3.7.6 Sada testov

Sada testov je zoznam testovacích skriptov, ktoré sa vykonajú v určenom poradí. Sada testov môže mať rovnako ako testovací prípad definované predpoklady.

3.7.7 Testovací záznam

Testovací záznam sú surové dáta získané vykonaním testovacích skriptov alebo sady testov. Konkrétne obsahuje všetky dôležité informácie o testoch, ako sú napríklad názov testovacieho skriptu, čas jeho začiatku a ukončenia, počet úspešne a neúspešne vykonaných testov a pod. Záznamy môžu tiež obsahovať obrazy testovanej aplikácie, odkazy na súbory a ďalšie druhy záznamov.

3.7.8 Výsledky testov

Výsledky testov sú v podstate spracované testovacie záznamy. Obsahujú hlavné výsledky a zanalyzované dáta.

4 Nástroje pre automatizované testovanie

Nástroje pre automatizované testovanie sú nástroje pre automatizáciu testov a tvorbu testovacích skriptov. Automatizáciu testov môžu umožňovať dvoma spôsobmi, a to nahrávaním vykonaných akcií používateľom v internetovom prehliadači a programovaním testov. Líšiť sa môžu okrem iného aj v tom, aké programovacie jazyky podporujú. Niektoré nástroje podporujú hneď niekoľko programovacích jazykov. Nástroje môžu fungovať ako samostatné programy, ako doplnok prehliadača alebo ako rozšírenie vo vývojom nástroji. Niektoré z najznámejších testovacích nástrojov v tejto kapitole popíšem bližšie.

4.1 UFT

UFT (Unified Functional Testing) [14] pôvodne nazývané QTP (QuickTest Professional) je testovací nástroj pre automatizáciu od spoločnosti Hewlett-Packard. Slúži na funkčné a regresné testovanie. Z internetových prehliadačov podporuje Chrome, Firefox, Internet Explorer a Safari.

Používa skriptovací programovací jazyk VBScript pre tvorbu testovacích skriptov. VBScript síce podporuje triedy, ale nepodporuje dedičnosť a polymorfizmus.

Využíva koncept **dátovo riadeného testovania**. Znamená to, že testovacie dáta, s ktorými testovací skript pracuje sú uložené v dátovom súbore (Excel súbor, XML, CSV, databáza a pod.). Tento súbor obsahuje vstupné testovacie dáta a môže tiež obsahovať očakávané výstupné dáta. Testy nie sú spúšťané napevno nastavenými dátami v zdrojovom kóde, ale sú od zdrojového kódu oddelené. Dáta tak môžeme rýchlo a bezproblémovo meniť.

Medzi jeho nevýhody patrí, že má užívateľské rozhranie len pre operačný systém Windows a je platený.

4.2 Selenium

Medzi najznámejšie testovacie nástroje nepochybne patrí **Selenium** [15]. Je to bezplatný nástroj s otvoreným zdrojovým kódom primárne určený k automatizácii webových prehliadačov. Používaný je na funkčné a regresné testovanie, neumožňuje záťažové testovanie ako niektoré platené nástroje. Podporuje dátovo riadené testovanie. Selenium umožňuje nahrávanie inštrukcií priamo z prehliadača a ich následné exportovanie do vybraného programovacieho jazyka. Čím vyniká oproti konkurencii je práve široká podpora programovacích jazykov a testovacích prostredí. Výsledný skript sa dá exportovať do jazykov Java, C#, Ruby, Python, PHP, JavaScript, Haskel, Perl, R, Ojective-C. Ďalšou výhodou je multiplatformovosť – je spustiteľný na Windowse, Linuxe a Mac OS. Pre Selenium existuje množstvo rozšírení, ktoré mu pridávajú ďalšiu funkcionalitu.

Selenium v základnej komponente s názvom Selenium IDE je doplnok prehliadača Mozilla Firefox. Je teda obmedzený len na tento webový prehliadač, čo znamená, že testy sa dajú nie len nahrávať ale aj spúšťať len nad Firefoxom. V praxi to môže byť veľkou nevýhodou, pretože rôzne prehliadače zobrazujú webový obsah odlišne, a tak je treba testovať na rôznych prehliadačoch. Túto nevýhodu riešia ďalšie komponenty a ovládače

pre prehliadače. Celkovo sú tak okrem Firefoxu podporované aj prehliadače Chrome, Safari, Opera, Internet Explorer a Microsoft Edge.

4.3 Test Studio

Test Studio [13] je sada testovacích nástrojov od spoločnosti Telerik slúžiacich k automatizovanému webovému testovaniu, testovaniu WPF aplikácií, záťažovému testovaniu, testovaniu mobilných aplikácií a API testovaniu. Umožňuje funkčné, regresné a záťažové testovanie. Okrem automatizovaného testovania podporuje aj manuálne testovanie. Z webových prehliadačov sú podporované Chrome, Firefox, Safari, Internet Explorer a Edge. Umožňuje testovanie REST API a Web API. Ďalej umožňuje testovanie mobilných aplikácií pre mobilné operačné systémy iOS a Android. Na tvorbu testov nám Test Studio dáva na výber medzi programovacími jazykmi C# a Visual Basic. Rovnako ako UFT a Selenium aj Test Studio využíva dátovo riadené testovanie.

Test Studio má 3 verzie produktu:

- Verzia pre funkčné a regresné testovanie – umožňuje webové a WPF testovanie. Okrem samostatnej aplikácie obsahuje aj rozšírenie do Visual Studia.
- Verzia pre záťažové testovanie.
- Plná verzia – kombinuje možnosti predchádzajúcich verzií a pridáva testovanie API a mobilných aplikácií.

Podpora produktu je na dobrej úrovni. Pár dní po registrácii produktu mi prišiel email (aj keď len automaticky generovaný) od samotného špecialistu z podpory s možnosťou smerovať naň prípadné dotazy ohľadom produktu. Dokumentácia je ľahko pochopiteľná a tiež prehľadná, s množstvom obrázkových ukážok. Súčasťou dokumentácie sú tiež kvalitne spracované videotutoriály. Samostatne sú v ponuke aj platené kurzy. Všimol som si ale nedostatku v dokumentácii a na oficiálnom fóre, ktorým je nefunkčnosť niektorých odkazov.

Nevýhodou Test Studia rovnako ako aj pri UFT je, že podporuje len operačný systém Windows a je platený.

4.4 Porovnanie testovacích nástrojov

Existuje množstvo nástrojov pre automatizované testovanie, každý má svoje výhody ale aj nevýhody.

V prípade, že okrem regresných testov potrebujeme robiť ďalšie druhy testov, akým je napr. záťažový, a naša firma si môže dovoliť investovať do plateného nástroja, tak je vhodné si vybrať komplexný nástroj akým je napr. Test Studio.

Naopak v prípade, že máme obmedzený rozpočet a vystačíme si s jednoúčelovým nástrojom, tak pre nás bude vhodný bezplatný ale overený nástroj akým je napr. Selenium. Podľa potreby ho jednoducho doplníme ďalšími nástrojmi.

5 Použité nástroje

V tejto kapitole uvádzam technológie, ktoré som využíval pri praktickej časti tejto práce.

5.1 Test Studio

Vo firme je používané už spomínané Test Studio, v ktorom som robil rovnako aj ja. Používal som ho na regresné a funkčné testovanie. Test Studio umožňuje otvoriť projekt v samostatnej aplikácii aj vo Visual Studiu pomocou rozšírenia. Stačí tak založiť projekt v samostatnej verzii, dať v aplikácii vygenerovať spustiteľný súbor pre Visual Studio, ten otvoriť a ďalej pracovať vo Visual Studiu. Nový projekt sa dá samozrejme založiť aj vo Visual Studiu.

5.2 Visual Studio

Visual Studio je vývojové prostredie od spoločnosti Microsoft. Test Studio k nemu pri inštalácii pridá vlastné rozšírenie. Väčšinu času som pracoval v samostatnej aplikácii Test Studia, z menšej časti aj vo verzii rozšírenia pre Visual Studio. Obidve majú skoro rovnaké vymoženosti, ale sú tu rozdiely.

Rozhodnutie, či pracovať v samostatnej aplikácii alebo vo Visual Studiu s rozšírením by som po vlastných skúsenostiach učinil nasledovne:

- Pre prácu s testovacími zoznamami je jediná možnosť použiť samostatnú aplikáciu, ktorá s nimi umožňuje plnohodnotnú prácu. Testovacie zoznamy pre Test Studio používajú príponu .aiilist, ktorú Visual Studio s rozšírením nepodporuje.
- K programovaniu testov je lepšie používať Visual Studio s rozšírením. To má narozdiel od samostatnej aplikácie lepšie doplňanie a podporu zdrojového kódu. Doplňovanie kódu je v samostatnej aplikácii možné, avšak len pre Test Studio API a nie je tak inteligentné ako vo Visual Studiu. Samostatná aplikácia nepodporuje ani vyhľadávanie v zdrojovom kóde kvôli čomu je v nej programovanie testov menej pohodlné.

5.3 TFS

TFS (celým názvom Team Foundation Server) je súbor nástrojov od spoločnosti Microsoft slúžiaci pre podporu spolupráce tímu pri vývoji softvéru. Podporuje kompletný životný cyklus vývoja softvéru. Umožňuje nám správu zdrojového kódu, požiadaviek, riadenie projektu, monitorovanie zmien, automatické zostavenie a testovanie kódu. Spoločnosť NetDirect s.r.o. využíva všetky jeho vymoženosti.

TFS je vyvíjané primárne pre Visual Studio a Eclipse, ale podporuje aj niektoré ďalšie vývojové prostredia. Integrované je aj v samostatnej verzii Test Studia pre správu verzií. V nej sú však podporované len základné funkcie pre správu verzií a neumožňuje pokročilejšie funkcie, ako napr. história verzií. Pre pokročilejšiu správu verzií som preto používal TFS vo Visual Studiu. TFS som používal aj pre tvorbu zostavení.

6 Inštalácia a konfigurácia použitého softvéru

Na pracovnom počítači som inštaloval chýbajúci potrebný softvér. Visual Studio som používal vo verzii 2015.

Inštalácia Test Studia je jednoduchá. Po registrácii a inštalácii som potreboval nastaviť TFS. Ak na zariadení nie je nainštalované Visual Studio alebo je nainštalovaná verzia novšia ako 2013, tak integrácia TFS s Test Studiom nebude fungovať. Pri pokuse o pripojenie na TFS sa zobrazí chybová hláška, ktorá hovorí, že nie je možné načítať modul riadenia kódu. Keďže používam Visual Studio 2015, tak sa mi táto chyba zobrazila a musel som tento problém vyriešiť. Riešením je nainštalovať Team Explorer. Je to potrebné preto, že Team Explorer vo verziách Visual Studia novších ako 2013 už nie je súčasťou produktu, a teda ho musíme nainštalovať osobitne.

Testy v Test Studiu som spúšťal na prehliadačoch Chrome a Firefox. Ku každému prehliadaču treba nainštalovať rozšírenia na nahrávanie a vykonávanie testov. Okrem toho treba každý prehliadač kalibrovať. Kalibrácia upraví nastavenia prehliadača tak, aby bol použiteľný pre Test Studio.

7 Testovaná webová aplikácia

Webová aplikácia, ktorú som testoval sa nazýva FastCentrik [16]. Je jedným z troch hlavných produktov firmy. Je to platforma pre tvorbu internetových obchodov. Vytvorená je v ASP.NET MVC, ako databázový systém využíva Microsoft SQL Server a k objektovo-relačnému mapovaniu využíva EntityFramework. Aplikácia používa webový framework AngularJS (verzia 1). V pláne je čoskoro prejsť na Angular 2. Spoločnosť si zakladá na udržiavaní najnovších verzií softvérových technológií používaných platformou. Aplikácia je rozdelená na frontendovú časť a administrátorské rozhranie, lokalizovaná je do piatich jazykov. Aktuálne je na platforme aktívnych viac než 1300 internetových obchodov.

FastCentrik funguje formou SaaS (Software as a Service). Znamená to, že aplikácia je zákazníkom dodávaná cez internet ako služba. SaaS odstraňuje potrebu inštalácie a prevádzky na vlastnom zariadení.

Podpora tretích strán je na vysokej úrovni. Platforma je napojiteľná na účtovné a ekonomické systémy Pohoda, Money S3, K2, Altus Vario a ďalšie. Ďalej umožňuje napojenie na porovnávače cien tovaru Heureka, Zboží, NejlepšíCeny a ďalšie. Podporuje integráciu s množstvom ďalších známych ale aj menej známych systémov tretích strán.

Aplikácia používa modulárny dizajn. Všetky moduly sa dajú kedykoľvek zapnúť alebo vypnúť v administrátorskom rozhraní. Používateľ s administrátorským prístupom si tak vyberie len tie moduly, ktoré uzná za vhodné.

V administrátorskom rozhraní je možnosť upravovať vyhľadávacie vlastnosti produktov a spolu s ďalšími technikami optimalizácie pre vyhľadávače používanými platformou sú tieto stránky na popredných pozíciách medzi vyhľadávačmi. Spolu s pomocou podpory tretích strán si tak internetové obchody zvyšujú počet zákazníkov.

Po objednaní FastCentriku dostane zákazník široký výber moderných šablón zdarma. Všetky šablóny sú responzívne a internetové obchodu sú tak použiteľné na zariadeniach so všetkými používanými rozlíšeniami. Tieto šablóny sa dajú upravovať v administrátorskom rozhraní.

8 Vytvorené testovacie scenáre

Vytvoril som celkovo 49 testovacích scenárov. Ich detailný výpis sa nachádza v prílohách. Každý testovací scenár ďalej obsahuje jeden alebo viac testovacích prípadov. Aby sa dalo medzi testovacími scenármi rýchlo vyhľadávať a kvôli sprehľadneniu som ich rozdelil do skupín podľa základnej štruktúry aplikácie. Skupiny s veľkým počtom testovacích scenárov som ďalej rozdelil na podskupiny. Používam ich pri testovaní, kde skupina alebo podskupina testov určuje oblasť aplikácie. Vďaka tomu sa môžeme zamerať na konkrétne oblasti aplikácie a overovať funkčnosť každej oblasti zvlášť.

8.1 Rozdelenie testovacích scenárov

1. **Prihlásenie**
V skupine prihlásenie sú testovacie scenáre súvisiace s prihlásením do frontendu aj administrátorského rozhrania. Niektoré z testov, ktoré tu patria sa často vykonávajú aj ako samostatné kroky v ďalších testoch, v ktorých je treba byť prihlásený.

2. **Kategórie**
V aplikácii sú produkty rozdelené do jednotlivých kategórií. Kategórie si môžeme predstaviť ako stromovú štruktúru, kde potomka kategórie nazývame podkategória. V tejto skupine sa nachádzajú testovacie scenáre súvisiace s kategóriami a podkategóriami.

3. **Produkty**
Skupina testovacích scenárov pre produkty zaberá široké spektrum funkcionality. Vytvoril som tak pre túto skupinu väčší počet testovacích scenárov.

Pre väčší počet scenárov v tejto skupine som skupinu rozdelil na nasledujúce podskupiny:

3.1. **Základná funkcionalita produktu** – v tejto podskupine produktov sa nachádzajú scenáre pre základnú funkcionalitu v administrátorskom rozhraní, ako sú pridanie, odstránenie, vyhľadávanie a úprava produktu.

3.2. **Cena produktu** – v tejto podskupine sú scenáre súvisiace s cenou produktu. Cena môže byť rôznych typov, ako sú katalógová, nastavená cena s DPH, nastavená cena bez DPH a výsledná cena po aplikovaní všetkých zliav. V tejto podskupine je niekoľko testovacích scenárov pre nastavenie, prepočítanie a overenie všetkých typov cien produktu.

3.3. **Zľavy produktu** – nachádzajú sa tu testovacie scenáre vzťahujúce sa na rôzne druhy zliav produktu a maximálny limit zľavy produktu.

3.4. **Výrobcovia produktu** – sú tu testy pre nastavenie a overenie výrobcu produktu.

3.5. **Súvisiace produkty** – patria tu testy pre súvisiace produkty daného produktu. Súvisiace produkty sa zobrazujú na frontende pri danom produkte. Sú to produkty, ktoré s vybraným produktom nejakým spôsobom súvisia. Pri kúpe produktu sa očakáva, že používateľ by mohol mať záujem aj o súvisiace produkty. Súvisiacimi produktmi

pre nejaký mobilný telefón sú napr. púzdro a ochranné sklo určené pre tento typ mobilného telefónu.

3.6. **Varianty produktu** – produkt môže mať rôzne varianty a byť tak na predaj napr. v rôznych farbách. V administrátorskom rozhraní sa dajú vytvoriť varianty pre daný produkt alebo použiť varianty z vopred pripravenej šablóny variant. Všetky testovacie scenáre súvisiace s variantami produktu spadajú do tejto podskupiny.

3.7. **Externé odkazy produktu** – každý produkt môže mať priradených viacero externých odkazov a videí. Funkcionalita s tým spojená je predmetom testovacích scenárov v tejto podskupine.

3.8. **Príznaky produktu** – produkt býva na frontende označený príznakmi, ako napr. novinka, zľava. Niektoré príznaky sú pridávané automaticky, vlastné si zas definujeme v administrátorskom rozhraní.

4. **Zľavy**

V skupine zliav sú testovacie scenáre týkajúce sa rôznych typov zliav. Tieto zľavy sa narozdiel od produktových zliav neaplikujú na jeden konkrétny produkt, ale sú aplikované na všetky produkty alebo na vybrané skupiny produktov. Okrem toho sa tu nachádzajú aj testovacie scenáre pre cenníky, čo sú zoznamy cien pre rôznych zákazníkov. Verní zákazníci tak môžu získať špeciálnu zľavu.

5. **Ostatné**

V tejto skupine sa nachádzajú všetky scenáre, ktoré nespadajú pod predchádzajúce skupiny. Sú tu testovacie scenáre, ktoré obsahujú pomocné testy a testy pre zaistenie predpokladov iných testov.

9 Automatizované testovanie v Test Studiu

Test Studio si pre každý test ukladá **DOM** (Document Object Model). Je to objektovo orientovaná reprezentácia XML, HTML alebo XHTML dokumentu. DOM je zobrazené v stromovej štruktúre, kde sa jednotlivé prvky nazývajú elementy. Element môže mať potomkov a predkov. Najbližší predok elementu sa nazýva rodič a najbližší potomkovia elementu sa nazývajú jeho dcérskymi potomkami. Element, ktorý nemá žiadnych predkov sa nazýva koreň.

V prípade, že test zlyhá, máme možnosť si zobraziť pôvodne nahraný DOM, na ktorom test fungoval správne a porovnať ho s DOM-om pri teste, ktorý zlyhal. Môžeme tak zistiť prečo test zlyhal v prípade, že sa nenašiel daný element. Navyše som testy vytváral tak, aby v prípade, že nejaký test zlyhá je z výsledku hneď jasné, kde chyba nastala.

Okrem DOM si Test Studio ukladá snímky prehliadača. Zobrazuje sa tak na porovnanie snímka prehliadača, pri ktorej test prebehol úspešne a aktuálna snímka prehliadača z testu, ktorý zlyhal.

V Test Studiu máme možnosť automatizované testy vytvárať nahrávaním interakcií s web stránkou alebo programovaním. Prvý test, ktorý som vytvoril bol vytvorený pomocou nahrávania, ďalšie testy som už vytváral len programovaním.

Výhody programovania testov:

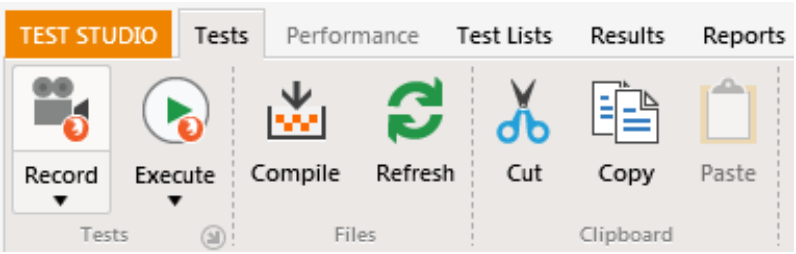
- Úplná kontrola nad tým, čo bude daný test vykonávať.
- Programátorské vymoženosti.

Výhody nahrávania testov:

- Rýchlosť tvorby testov.
- Čiastočnou výhodou je jeho preveditelnosť do zdrojového kódu. Nahraný test síce nemá všetky programátorské vymoženosti ako naprogramovaný, ale Test Studio ho dokáže previesť do zdrojového kódu. Následne s ním pracujeme ako s naprogramovaným testom.

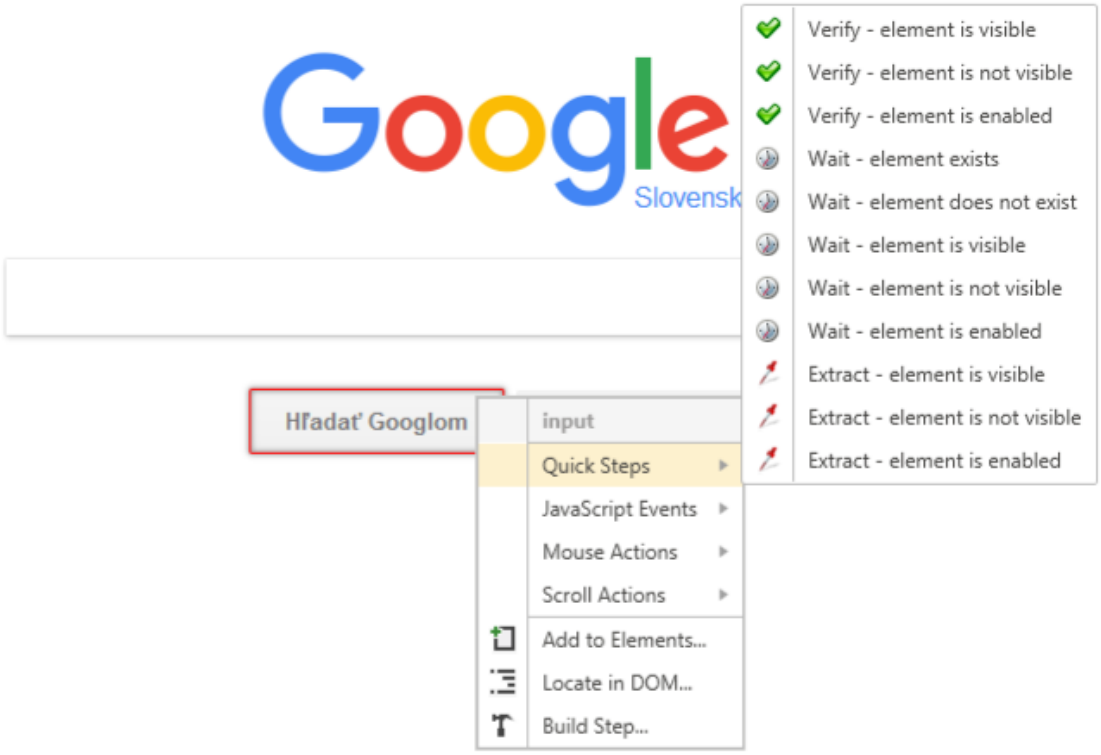
9.1 Ukážka nahrávania testu

Ak chceme vytvoriť test nahrávaním inštrukcií, tak klikneme na tlačidlo „Record“, viď obrázok 3. Následne vyberieme, aký typ testu chceme nahrávať. Na výber máme webový test, záťažový test, WPF test a manuálny test. Vyberieme webový test a zadáme mu názov.



Obrázok 3: Výber nahrávania testu z ponuky

Následne sa objaví okno, ktoré po nás bude požadovať URL adresu, na ktorej má nahrávanie začať. Po jej zadaní sa otvorí webový prehliadač na zadanej URL adrese a môžme začať nahrávať. Keď je prehliadač v stave nahrávania, interakcia s prehliadačom sa zaznamenáva a ukladá priamo do Test Studia. Ak kurzor myši premiestnime nad nejaký element stránky, zobrazí sa ponuka (viď obrázok 4). Na výber máme vyvolanie rôznych JavaScriptových udalostí, akcie myši a posúvania sa na stránke. Ďalej tu máme tzv. rýchle kroky, pomocou ktorých môžeme na element použiť rôzne overenia, čakania alebo si môžeme nejakú jeho vlastnosť extrahovať a použiť ju v niektorom z ďalších krokov. Nevýhodou ale pri takejto manipulácii s elementami je, že Test Studio si k nemu sám vytvorí dotaz pre nájdenie tohto elementu v DOM. Tento dotaz často nie je taký, aký by sme ho chceli mať, preto je tu ešte možnosť ho ručne zmeniť v Test Studiu. Osobne to považujem za nepohodlné a preferoval by som, keby bola možnosť vytvorenia si dotazu už v ponuke v prehliadači. Rýchlejšie a pohodlnejšie je pre mňa automatizované testy programovať.



Obrázok 4: Ponuka pre element pri nahrávaní testu

9.2 Testovacie dáta

Test Studio používa dátovo riadené testovanie. Dáta môžu byť uložené v rôznych typoch súborov (XML, Excel), v databáze alebo ako lokálne dáta. Ja som na ukladanie dát používal XML súbor a štrukturoval som ho podľa kategórií funkcionality. Vďaka dátovo riadenému testovaniu sa dáta dajú jednoducho meniť. Pri testovaní som každý test spúšťal opakovane nad rôznymi dátami.

Keď máme pripravený element testovacích dát, ktorý budeme v teste používať, potrebujeme ho danému testu ešte priradiť, aby Test Studio vedelo, s ktorým elementom má pracovať. Vyberieme „zviazať s dátami“ a zobrazí sa zoznam všetkých unikátnych značiek nachádzajúcich sa v dátovom súbore. Vyberieme ten, ktorý potrebujeme a zmeny uložíme. Podľa toho koľkokrát sa daná značka v súbore nachádza, toľkokrát sa daný test opakovane spustí s dátami priradenými daným elementom. Dáta z viacerých XML elementov do jedného testu dostaneme pomocou podtestov.

9.3 Problémy pri automatizácii testov

Pri tvorbe automatizovaných testov som sa stretol s niekoľkými väčšími a menšími problémami. Vyskytli sa z viacerých dôvodov, tými najčastejšími sú samotný charakter aplikácie a to, že Test Studio potrebnú funkčnosť štandardne nepodporuje. Každý problém sa dal nejakým spôsobom vyriešiť. Problémy, s ktorými som sa počas tvorby automatizovaných testov stretol a s nimi súvisiace odporúčania popíšem bližšie. Popíšem aj spôsoby riešenia týchto problémov a ako som ich riešil ja.

9.3.1 Automatizované kliknutie nemá požadovaný efekt

Existujú elementy, ktoré nereagujú na štandardné kliknutie používané Test Studiom. Môže to byť z rôznych dôvodov, napr. preto, že sa nevyvolala JavaScript udalosť. Test Studio tento nedostatok preto rieši tým, že má rôzne typy kliknutí pre rôzne účely.

Štandardná udalosť `Click()` je spustená priamo nad DOM-om prehliadača. Tento typ kliknutia vo väčšine prípadov funguje správne, ale nevyvoláva JavaScriptovské udalosti priradené danému elementu.

Okrem štandardného sa dá vykonať kliknutie, ktoré dôverne simuluje skutočné kliknutie užívateľom a to tak, že presunie kurzor myši nad daný element a následne vykoná kliknutie. Vyvoláva sa metódou `MouseClick()` a funguje tak, ako keby sme na element klikli myšou my samy, a preto funguje vždy. Tento typ kliknutia ale z vlastnej skúsenosti neodporúčam kvôli presúvaniu kurzoru myši počas testovania, čo je z viacerých dôvodov nepraktické, a preto som ho používal len v nevyhnutných prípadoch.

Ďalšou možnosťou je vyvolať kliknutie pomocou JavaScriptu. V tomto prípade sa správne vyvolajú JavaScriptovské udalosti.

9.3.2 Štandardné automatizované zadanie vstupu nemá požadovaný efekt

Pri niektorých menej štandardných vstupoch, akými sú napr. WYSIWYG editory sa nedá zadávať hodnota základným zadaním vstupu v Test Studiu. To funguje len pre základné HTML5 vstupy. Riešením je simulovať písanie textu. Robí sa to tak, že nastavíme kurzor na písanie na daný vstup a simulujeme písanie textu z klávesnice. Slúži na to príkaz `Manager.Desktop.KeyBoard.TypeText("text", 10, 10, true);`. Prvým parametrom je text, ktorý sa bude simulovať, druhý parameter udáva čas čakania medzi jednotlivými stlačeniami klávesov v milisekundách, tretí parameter udáva ako dlho má byť jeden kláves stlačený v milisekundách a posledný parameter hovorí, či má byť povolený Unicode (štandard pre reprezentáciu znakov). Druhý a tretí parameter by mali mať pokiaľ možno čo najnižšie hodnoty, aby sa skrátil čas behu testov.

9.3.3 Vyhľadávanie elementov v DOM

Pre vyhľadávanie elementov v DOM sa dá v Test Studiu použiť viacero metód vyhľadávania, akými sú vyhľadávanie pomocou identifikátora, pomocou názvu značky a jej poradia, pomocou hodnôt atribútov a za použitia XPath. Ja som najčastejšie používal

XPath, pretože sa ním dajú simulovať ostatné metódy a zároveň z nich ponúka najpokročilejšie možnosti vyhľadávania v DOM.

Ideálne je vyhľadávať elementy dotazom, ktorý je čo najmenej závislý na samotnom DOM. DOM sa totiž môže často meniť a nemusí tak byť rovnaký po opätovnom spúšťaní testu. Taktiež sa môže meniť každou novou verziou aplikácie. Nie je preto vhodné používať dotaz, ktorý vyhľadáva element s pomocou fixného poradia medzi elementmi, ako napr. `/html/section[1]/div[4]/a[2]`.

Ak element nemá nastavený identifikátor, tak je najlepšie vyhľadávať elementy pomocou HTML značiek, jeho predkov, atribútov udávajúcich nejakú jeho hodnotu (atribút `value`) a ak je možnosť, tak môžeme použiť atribút, ktorý odkazuje na účel tohto elementu (napr. class="logout"). Riešením ale nie je odkazovať sa na triedu (atribút `class`), ktorá je viazaná na CSS štýly, pretože sa môže často meniť.

9.3.4 Práca s viacerými záložkami v prehliadači naraz

Test Studio nedokáže pracovať s viac než jednou otvorenou stránkou v prehliadači naraz. V prípade, že kliknutie na odkaz by nám otvoril stránku na ďalšej záložke, musíme takéto riešenie obísť. Riešil som to tak, že namiesto kliknutia na takýto odkaz som sa premiestnil na danú URL adresu.

Podporovaná je však práca vo viacerých prehliadačoch naraz. V každom z nich máme znova možnosť pracovať len s jednou stránkou naraz. Dá sa tak napr. simulovať viac používateľov naraz. S každým prehliadačom sa dá pracovať rovnako ako keď pracujeme len s jedným a získané dáta medzi prehliadačmi sa dajú zdieľať. Namiesto tejto funkcie by som osobne prijal možnosť práce s viacerými záložkami, keďže možnosť výberu prehliadača máme už pri samotnom spúšťaní testov.

```
// Spustenie IE
Manager.LaunchNewBrowser(BrowserType.InternetExplorer, true);

// Ulozime si prehliadac, prehliadac je po spusteni aktivny
Browser ie = Manager.ActiveBrowser;

// Spustenie Firefoxu
Manager.LaunchNewBrowser(BrowserType.FireFox, true);

// Ulozime si prehliadac, prehliadac je po spusteni aktivny
Browser ff = Manager.ActiveBrowser;

// Spustenie dalsej instancie Firefoxu
Manager.LaunchNewBrowser(BrowserType.FireFox, true);

// Spustene prehliadace su ulozene v kolekcii Browsers
Browser ff2 = Manager.Browsers[Manager.Browsers.Count - 1];

// Praca s instanciami prehliadacov
ie.NavigateTo("https://www.google.com");
ff.NavigateTo("https://www.google.com");
```

```
ff2.NavigateTo("http://www.telerik.com");
ie.NavigateTo("http://www.telerik.com");

// Pomocou vlastnosti Window mozeme pristupovat k oknu prehliadaca
ie.Window.SetFocus();

// Zatvorenie prehliadaca
ff.Close();
```

Výpis 1: Ukážka práce s viacerými prehliadačmi

9.3.5 Odlišné zobrazenie v jednotlivých prehliadačoch

Pri formulároch sa v Google Chrome zobrazujú odkazy na časti formulára, ale vo Firefoxe sa tieto odkazy nezobrazujú a nenachádzajú sa ani v DOM. Tento rozdiel v prehliadačoch ovplyvňoval XPath dotaz, a tak som to musel riešiť osobitne pre prípad, že sa v DOM nachádzajú tieto odkazy a osobitne pre prípad, že sa tam odkazy nenachádzajú. Podobné prípady, kde sa nezhodujú zobrazenia v prehliadačoch sa môžu vyskytovať pomerne často. Treba preto testovať na rôznych prehliadačoch a nájsť optimálne riešenie problému. Ideálne by mali byť testy dostatočne obecné na to, aby fungovali na všetkých prehliadačoch. Ak to však nejde, tak treba problém riešiť osobitne pre jednotlivé prehliadače ako tomu bolo v mojom prípade.

9.4 Pomocné metódy

Vytvoril som si pomocné metódy, ktoré som d'alej v zdrojovom kóde využíval. Jednou z nich je metóda pre získanie desatinného čísla z textu. Túto metódu som vytvoril tak, aby dokázala pracovať s textom obsahujúcim peňažnú sumu v rôznych menách a s rôznymi oddeľovačmi. Využívam ju vždy pri práci s cenami, obzvlášť v testoch patriacich do skupín produktov a zliav.

```
public static double TextToDouble(string text)
{
   Assert.IsFalse(string.IsNullOrWhiteSpace(text));
   char[] allowedCharsInText = { '.', ',', '-' };
   char[] charArray = text.Where(c => (char.IsDigit(c) ||
       allowedCharsInText.Contains(c))).ToArray();
   text = new string(charArray);
   text = text.Replace(',', '.');

   double result;
   double.TryParse(text, NumberStyles.Any, CultureInfo.InvariantCulture,
       out result);
   return result;
}
```

Výpis 2: Pomocná metóda na získanie čísla z textu

Ďalšia z pomocných metód súvisí s tým, že Test Studio dokáže pracovať len s jednou základnou URL. Ja som však potreboval pracovať s dvoma základnými URL, a to pre frontend a administrátorské rozhranie. Vytvoril som si na to jednoduchú, ale účinnú metódu, ktorá uprostred testu základné URL dočasne zmení. Používam ju pred samotným testom na prihlásenie do administrátorského rozhrania. Metóda zo základného URL získa URL pre administrátorské rozhranie.

```
public string GetAdminUrl()
{
   // ziskame zakladne URL
   string baseUrl = Settings.Current.Web.BaseUrl;

   string splitSeparator = "://";

   string[] splittedBaseUrl = new string[2];
   splittedBaseUrl = baseUrl.Split(new string[]{ splitSeparator },
       StringSplitOptions.None);

   // pridame prefix
   string prefix = "admin.";
   return splittedBaseUrl[0] + splitSeparator + prefix + splittedBaseUrl
       [1];
}
```

Výpis 3: Pomocná metóda pre získanie URL do administrátorského rozhrania zo základného URL

9.5 Automatizácia testovacieho prípadu

Spravil som testovacie skripty pre celkovo 49 testovacích prípadov. Testovacie skripty a testovacie prípady som nie vždy vytváral formou jedna k jednej, pretože by tak beh testov trval dlhšie a množstvo krokov by sa muselo opakovať. Jeden testovací skript preto môže zahŕňať viacero testovacích prípadov. Zameral som sa na znovupoužiteľnosť testovacích skriptov. Takéto testovacie skripty sa dajú použiť ako samostatný krok v iných testovacích skriptoch, čím sa stávajú ľahšie udržovateľnými.

Pre vysvetlenie si ako príklad zoberme testovacie prípady pre nastavenie ceny produktu v administrátorskom rozhraní a overenie ceny produktu na frontende. V prvom testovacom prípade nastavíme cenu produktu, overíme správnosť nastavenia ceny a uložíme si aktuálne DPH v percentách (je viditeľné len v administrátorskom rozhraní). Toto DPH je potom použité na overenie ceny na frontende v druhom testovacom prípade. Takýmto naviazaním testovacích skriptov na seba ušetríme množstvo krokov v testovacom skripte, pretože pri overení ceny sa nemusíme vracať znova do administrátorského rozhrania, aby sme zistili aktuálne DPH.

9.5.1 Testovací prípad

Popíšeme si testovací prípad a postup automatizácie testu k nemu. Pre jednoduchosť pochopenia som vybral kratší testovací prípad.

Názov: ADMIN – Pridanie súvisiacich produktov

ID: #32

Účel:

- Pridanie súvisiacich produktov k danému produktu

Predpoklady:

- Užívateľ je registrovaný
- Existuje produkt s názvom Data["name"]
- Existuje aspoň jeden súvisiaci produkt
- Súvisiace produkty existujú
- Súvisiace produkty sú viditeľné na FE
- Daný produkt je viditeľný na FE
- Na ADMIN v "Marketing" -> "Personalizace", je zapnuté "Související zboží" a nastavené na "Pouze ručně vybrané"

Test. data: fastcentrik/products/product/relatedProducts/relatedProduct

Test. skript: Product/Backend/RelatedProducts/AddRelatedProductTest

Kroky:

1. Vykonáme test ADMIN - Odstránenie všetkých súvisiacich produktov
    a. Očakávaný výsledok
        i. Neexistuje žiadny súvisiaci produkt
        ii. Existuje tlačidlo "Přidat produkt"
2. Pre každý súvisiaci produkt z test. data
    a. Klikneme na tlačidlo "Přidat produkt"
        i. Očakávaný výsledok
            1. Zobrazí sa práve jedno okno výberu produktu
            2. V okne existuje vstup "Vyhledávaní produktu"
            3. V okne existuje tlačidlo "Přidat vybrané produkty"
    b. Nastavíme "Vyhledávaní produktu" na Data["code"]
        i. Očakávaný výsledok
            1. Vyhľadávaný produkt bol nájdený
    c. Klikneme na nájdený produkt
        i. Očakávaný výsledok
            1. Existuje tlačidlo "Přidat vybrané produkty" a je aktívne
    d. Klikneme tlačidlo "Přidať vybrané produkty"
        i. Očakávaný výsledok
            1. Existuje tlačidlo "Uložit"
            2. Tlačidlo "Uložit" je aktívne
3. Klikneme na tlačidlo "Uložit"
    a. Očakávaný výsledok
        i. Zobrazí sa okno o úspešnom uložení
4. Počkáme pokým sa zavrie okno s oznámením o úspešnom uložení
    a. Očakávaný výsledok
        i. Oznamovacie okno sa zavrie
        ii. Na ADMIN sa u daného produktu sa zobrazujú pridané súvisiace produkty

Obrázok 5: Testovací prípad – ADMIN - Pridanie súvisiacich produktov

V testovacom prípade na obrázku 5 vidíme ako prvý **názov** testovacieho prípadu, ktorý je ADMIN - Pridanie súvisiacich produktov. Vo firme je zaužívané používať ako prefixy v názve FE a ADMIN podľa toho, či sa test vykonáva na frontende alebo na administrátorskom rozhraní. Názov by mal byť skrátenou verziou účelu.

ID testovacieho prípadu je unikátnym identifikátorom testovacieho prípadu. Môžeme sa pomocou neho odkazovať na konkrétny testovací prípad.

Účel je popis funkcionality, ktorú testovací prípad testuje. V tomto prípade budeme testovať pridanie súvisiacich produktov k danému produktu.

Predpoklady testovacieho prípadu určujú aké podmienky musia byť splnené, aby sme test mohli vykonať. Chceme dosiahnuť, aby ich bolo čo najmenej a test by tak mohol prebehnúť bez nášho zásahu. V tomto prípade väčšinu predpokladov splní testovací zoznam, v ktorom sa testy vykonávajú postupne a splnia niektoré jeho predpoklady.

Predpoklad „existuje produkt s názvom Data["name"]“ hovorí, že produkt, ktorý máme definovaný v testovacích dátach existuje. Data["name"] je zápis, ktorý sa potom v testovacích dátach odkazuje na názov produktu.

Niektoré predpoklady musia byť splnené pri výbere testovacích dát. Pre predpoklady „súvisiace produkty existujú“ a „súvisiace produkty sú viditeľné na FE“ sa teda vyberú také testovacie dáta, ktoré tieto predpoklady spĺňajú.

Časť **test. dáta** určuje cestu k testovacím dátam v XML súbore. Ak sa potom v rámci testovacieho prípadu odkazujeme na Data["name"], tak hľadaný element s názvom name nájdeme pod touto cestou.

Test. skript udáva cestu k testovaciemu skriptu v Test Studiu. Rýchlo tak dokážeme nájsť testovací skript prislúchajúci testovaciemu prípadu.

Kroky sú jednotlivé kroky, ktoré sa budú po spustení testu vykonávať. Môžeme z nich vyčítať stručný popis toho čo bude test vykonávať.

Povedzme si v úplnej stručnosti čo test vykoná. Najskôr nás test prihlási do administrátorského rozhrania, a dostaneme sa k súvisiacim produktom vybraného produktu. Ak nejaké súvisiace produkty existujú, tak sa všetky odstránia. Následne vytvoríme nové súvisiace produkty, uložíme ich. Test končí overením týchto produktov.

Po vykonaní tohto testu v ďalšom nadväzujúcom teste overíme, či sa zhodujú vytvorené súvisiace produkty v adminstrátorskom rozhraní a na frontende.

Jednotlivé kroky pridania a overenia súvisiach produktov si popíšeme bližšie.

Prvým krokom je vykonanie testu s názvom ADMIN - Odstránenie všetkých súvisiacich produktov. Tu máme možnosť vidieť spomínanú znovupoužiteľnosť testov. Keďže chceme overiť správnosť pridávania súvisiacich produktov, odstránime najskôr existujúce produkty. Vykoná to za nás tento už existujúci test. V tomto prípade slúži ako tzv. **podtest**, no dá sa spustiť aj samostatne. Podtest nás najskôr prihlási do administrátorského rozhrania (ďalší podtest). V prípade, že je prihlásenie neúspešné zlyhá aktuálny podtest a teda aj rodičovský test. Ak je prihlásenie úspešné, tak pokračuje na produkt s názvom, ktorý získame z testovacích dát. Tento produkt musí už pred spustením podtestu existovať. Rovnaký predpoklad máme aj pri rodičovskom teste. Vidíme tak, že predpoklady podtestu bývajú často aj predpokladmi rodičovského testu.

Keď sa nachádzame na produkte, klikneme na záložku s alternatívnymi a súvisiacimi produktmi. V cykle potom vyhľadávame tlačidlá pre odstránenie existujúcich súvisiacich produktov. Ak je takéto tlačidlo nájdené, tak sa naň klikne a vyhľadáva sa znova. Ak nájdené nie je, cyklus sa ukončí. Podtest preto správne prebehne v poriadku aj v prípade, že nebudú nájdené žiadne súvisiace produkty.

Po každom kliknutí na odstránenie je potrebné počkať, pokým sa produkt skutočne odstráni. V opačnom prípade by mohlo dôjsť k ďalšiemu kliknutiu príliš skoro. Používa sa na to niekoľko metód tzv. čakania. Pri niektorých býva vstupným parametrom presná, prípadne maximálna doba čakania. Tá sa vždy udáva v milisekundách.

Ako prvé použijeme čakanie, pokým nebude prehliadač v stave „pripravený“. V takomto stave je vtedy, keď je stránka plne načítaná so všetkými ďalšími zdrojmi (obrázky, CSS, JavaScript, atď.).

Po tomto čakaní je ešte potrebné čakať, pokým sa dokončí AJAX požiadavka.

Čakanie na AJAX však nemusí fungovať vždy, a preto sa ešte používa uspanie vlákna. Pokiaľ je to možné, je dobré sa tomuto čakaniu vyhýbať, aby sme zbytočne nepredlžovali beh testu.

Po čakaní by sme mali obnoviť DOM. Je to preto, že Test Studio si ukladá používanú časť DOM stromu. Mohlo by sa tak stať, že by sa Test Studio pokúšalo pracovať s elementom, ktorý už v DOM neexistuje.

Pre prácu s webovým prehliadačom sa používa inštancia `ActiveBrowser`. Pomocou nej pristupujeme aj ku niektorým spomínaným metódam čakania a obnoveniu DOM stromu.

```
// Cakanie
ActiveBrowser.WaitUntilReady();
ActiveBrowser.WaitForAjax(3000);
System.Threading.Thread.Sleep(250);

// Obnovenie DOM stromu
ActiveBrowser.RefreshDomTree();
```

Výpis 4: Čakanie a obnovenie DOM stromu

Na konci prvého kroku overíme, že produkt nemá žiadne súvisiace produkty. Najskôr si uložíme výraz pre nájdenie HTML elementu. Takýto element obsahuje len atribút ng-click, a tak za použitia vyhľadávacieho výrazu s pomocou atribútu ng-click sa pokúsime tento element nájsť. Následne overíme, či element existuje. Ak neexistuje, znamená to, že neexistujú žiadne súvisiace produkty a krok prebehne v poriadku. V opačnom prípade test zlyhá s chybovou hláškou, že výsledkom overenia `Assert.IsNull(removeButton);` bola nepravda namiesto očakávanej pravdy.

```
HtmlFindExpression removeButtonFind = new HtmlFindExpression("xpath=(//a[
    contains(@ng-click, 'remove') and contains(@ng-click, 'commodity')])
    [1]");
```

```
// Pokusime sa element najst
Element removeButton = ActiveBrowser.Find.ByExpression(removeButtonFind);

// Overime, ze element neexistuje
Assert.IsNull(removeButton);
```

Výpis 5: Overenie, že element neexistuje

V **druhom kroku** pridáme všetky súvisiace produkty. V súbore s testovacími dátami sú uložené kódy týchto produktov, pomocou ktorých ich budeme vyhľadávať.

Pre každý súvisiaci produkt najskôr klikneme na tlačidlo „Přidat produkt“. Otvorí sa okno pre pridanie produktu. Tu som pridal overenie, že existuje práve jedno takéto okno. V ňom pomocou kódu z testovacích dát vyhľadáme produkt a vyberieme ho. Vybraný produkt potom pridáme medzi súvisiace produkty.

Celý tento proces druhého kroku sa vykonáva ako jeden podtest. Test Studio za nás rieši, že ak máme medzi testovacími dátami viac XML elementov, na ktoré je náš test naviazaný, tak sa test vykoná pre každý z nich. Platí to aj pre podtesty. Stačilo tak naprogramovať pridanie jedného súvisiaceho produktu a do testovacích dát pridať kódy pre potrebný počet súvisiacich produktov.

```
...
<relatedProducts>
  <relatedProduct>
    <code>529402</code>
  </relatedProduct>
  <relatedProduct>
    <code>681420</code>
  </relatedProduct>
  <relatedProduct>
    <code>227935</code>
  </relatedProduct>
</relatedProducts>
...
```

Výpis 6: Ukážka testovacích dát súvisiacich produktov

Pripomeňme si, že test je naviazaný na testovacie dáta XML elementu `relatedProduct`. Pre testovacie dáta z Výpis 6 sa potom test vykoná najskôr pre kód 529402, potom pre 681420 a nakoniec pre 227935.

Keď už máme všetky produkty pridané, tak si ešte uložíme ich názvy, pretože ich budeme potrebovať pri overovaní na frontende. Budeme teda potrebovať tieto názvy zdieľať naprieč krokmi testu. V Test Studiu sa to robí pomocou tzv. extrakcie. V jednom kroku testu nastavíme extrahovanú hodnotu pomocou metódy `SetExtractedValue`, ktorá má ako parametre názov, pomocou ktorého budeme k hodnote pristupovať a parameter samotnej hodnoty. V ďalších krokoch testu túto hodnotu môžeme získať pomocou metódy `GetExtractedValue` s parametrom nastaveného názvu hodnoty.

```
// List nazvov produktov
var relatedProductNames = new List<string>();

// Vyraz, podla ktoreho najdeme suvisiace produkty
string relatedProductExpr = "xpath=(//*[@id='
    commodity_relatedsection_relatedcommodities']//a[@ng-bind='commodity.
    Name']/parent::*)";

Element relatedProductEl;

int i = 1;
while (true)
{
   relatedProductEl = Find.ByExpression(relatedProductExpr + "[" + i + "]"
       );

   if (relatedProductEl == null)
   {
      break;
   }

   // Nazov produktu je vo vnutri elementu
   relatedProductNames.Add(relatedProductEl.InnerText);

   i++;
}

SetExtractedValue("relatedProductNames", relatedProductNames);
```

Výpis 7: Ukážka získania názvov súvisiacich produktov

Keď máme názvy uložené, môžeme všetky vykonané zmeny produktu uložiť. Overíme, že tlačidlo „Uložit“ existuje a je aktívne. Aktívne je vtedy, keď boli vykonané zmeny, ktoré sa dajú uložiť.

V **treťom kroku** klikneme na tlačidlo uložiť. Malo by sa zobraziť okno o úspešnom uložení. Overíme, či sa tak naozaj stalo. Pre zjednodušenie tento krok zlúčime so **štvrtým krokom**. V ňom čakáme, pokým sa oznamovacie okno samo zatvorí. Ak sa tak nestane do ôsmich sekúnd, test zlyhá.

```
HtmlFindExpression dialogFind = new HtmlFindExpression("xpath=(//div[
    contains(@class, 'modal-content')])[1]");

ActiveBrowser.WaitForElement(dialogFind, 1000, false);
HtmlControl dialog = ActiveBrowser.Find.ByExpression<HtmlControl>(
    dialogFind);
dialog.Wait.ForExistsNot(8000);
```

Výpis 8: Čakanie na otvorenie a zatvorenie oznamovacieho okna

Tým je test na administrátorskom rozhraní hotový. Pri každej zmene v administrátorskom rozhraní je ale treba overiť zmeny aj na frontende. Náš test preto nadväzuje na test **FE – Overenie súvisiacich produktov**. Ukážeme si stručný postup aj pre tento test.

V **prvých troch krokoch** sa prihlásime na frontende, dostaneme sa do kategórie nastavenej v testovacích dátach a nájdeme v nej náš produkt.

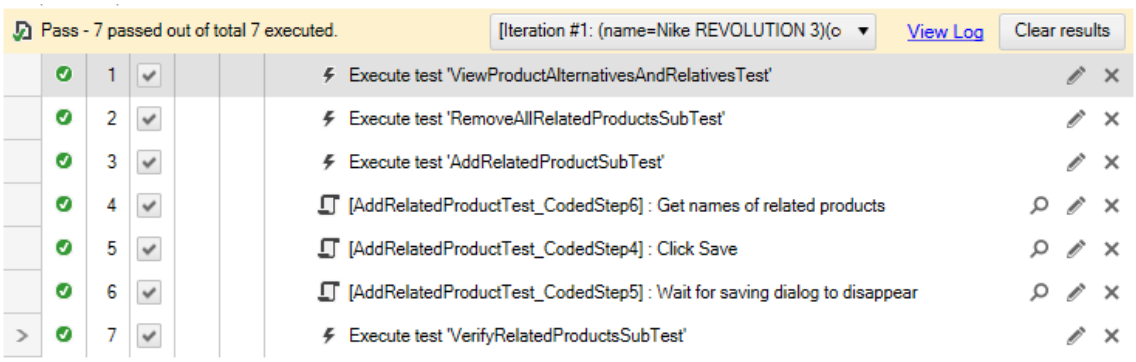
V **štvrtom kroku** budeme pri produkte overovať názvy súvisiacich produktov. Najskôr si získame extrahovaný zoznam názvov súvisiacich produktov. V cykle potom pre každý názov v zozname overíme, či sa na stránke naozaj nachádza. Ak áno, tak toto meno zo zoznamu odstránime, inak test skončí neúspešne. Cyklus sa opakuje, pokým zoznam nie je prázdny.

```
// Ziskame zoznam nazvov
var expectedRelatedProductNames = new List<string>((List<string>)
    GetExtractedValue("relatedProductNames"));
Assert.IsNotNull(expectedRelatedProductNames);

string relatedProductNameExpr = "xpath=(//*[@id='CommodityRelated'][1]//*[
    contains(@class, 'title')])";

int i = 1;
while (expectedRelatedProductNames.Any())
{
   Element relatedProductNameEl = ActiveBrowser.WaitForElement(3000,
       relatedProductNameExpr + "[" + i + "]");

   Assert.IsNotNull(relatedProductNameEl);

   // Posunieme sa v prehliadaci na najdeny element
   Actions.ScrollToVisible(relatedProductNameEl);

   string actualRelatedProductName = relatedProductNameEl.InnerText;

   // Overime, ze nazov produktu je v zozname, a potom ho z neho
       odstranime
   Assert.IsTrue(expectedRelatedProductNames.Contains(
       actualRelatedProductName),
      "Ocakavany suvisiaci produkt sa na stranke nenachadza");
   expectedRelatedProductNames.Remove(actualRelatedProductName);

   ActiveBrowser.WaitUntilReady();
   ActiveBrowser.WaitForAjax(2000);
   ActiveBrowser.RefreshDomTree();

   i++;
}
```

Výpis 9: Overenie názvov súvisiacich produktov na frontende

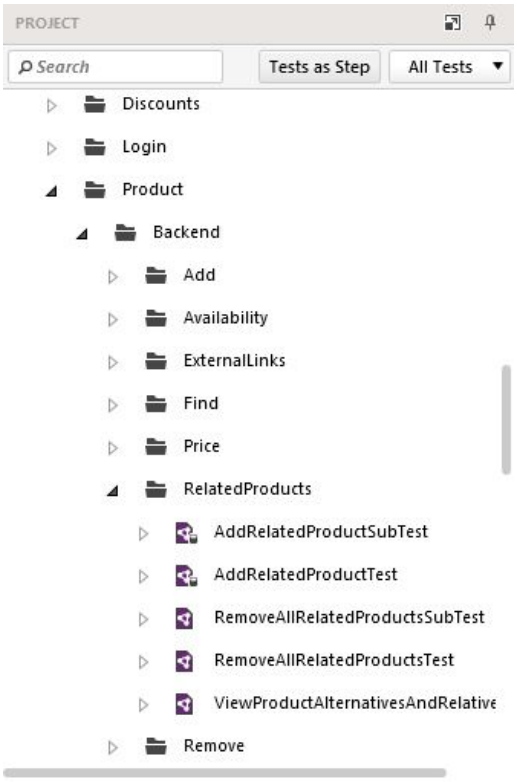
Celý test je po tomto kroku hotový a pripravený na spustenie. Funkcionalitu spus-

tíme nad rôznymi dátami. Test prebehol vždy úspešne pre dáta, ktoré spĺňajú požiadavky testu. Po spustení testu z obrázku 6 vidíme, že test prebehol úspešne (7 z celkových 7 krokov uspelo). Môžeme preto povedať, že časť funkcionality, ktorú sme otestovali funguje správne.



Obrázok 6: Ukážka krokov testu v Test Studiu

Na obrázku 7 je vidieť časť stromovej štruktúry testov v Test Studiu s pohľadom na súbory testov pre súvisiace produkty.



Obrázok 7: Ukážka časti stromovej štruktúry testov pre súvisiace produkty v Test Studiu

10 Testovacie zoznamy

Vytvoril som testovacie zoznamy, kde každý zoznam obsahuje súvisiace a ovplyvňujúce sa testy. Jednotlivé testy nie sú spúšťané v izolovanom prostredí, a tak sa navzájom ovplyvňujú, čo znamená, že na postupnosti testov záleží. Usporiadal som testy v logickom poradí tak, aby test A, ktorý spĺňa predpoklady testu B bol vykonaný pred testom B. Všetky predpoklady nadväzujúcich testov sú tak splnené. Okrem toho som dodržiaval pravidlo, aby zoznam testov ako celok nemal žiadne prípadne mal čo najmenšie požiadavky a bol tak spustiteľný kedykoľvek aj na vzdialenom stroji bez zásahu človeka.

10.1 Testovacie zoznamy v Test Studiu

V Test Studiu sa dajú do testovacích zoznamov pridávať buď manuálne, automatizované alebo výkonnostné testy. Tieto typy testov nejde v jednom testovacom zozname kombinovať. Na začiatku si pri tvorbe testovacieho zoznamu vyberieme názov testovacieho zoznamu a typ testov, ktoré doňho budeme pridávať. Potom pridáme samotné testy a nastavíme vlastnosti testovacieho zoznamu.

Mnou vytvorené testovacie zoznamy sa budú spúšťať osobitne podľa toho, ktorú časť funkcionality je v danej chvíli potrebné otestovať. Budú spúšťané ako na vzdialenom tak aj na miestnom počítači.

Test Studio umožňuje vytvárať dva typy testovacích zoznamov – statické a dynamické.

10.1.1 Statický testovací zoznam

Statický testovací zoznam je taký testovací zoznam, do ktorého si pridávame testy samy pomocou výberu zo zoznamu všetkých testov.

10.1.2 Dynamický testovací zoznam

Dynamický testovací zoznam nám umožňuje definovať pravidlá, podľa ktorých budú testy do zoznamu pridávané automaticky Test Studiom. Tieto pravidlá sa odkazujú na vlastnosti testov, ktorými sú názov, priorita, cesta, vlastník a ďalšie vlastnosti, ktoré si môžeme samy definovať. Pre tieto vlastnosti potom určujeme podmienky ako napr. či obsahujú nami definovaný výraz alebo týmto výrazom začínajú, končia atď. Týchto pravidiel sa dá pre jeden testovací zoznam definovať viac. Do testovacieho zoznamu sa potom automaticky pridávajú tie testy, ktoré spĺňajú všetky nami definované pravidlá. Dokážeme si tak napr. definovať testovací zoznam, do ktorého budú automaticky pridávané testy, ktorých cesta začína na `Category\Backend\Duplicate` a názov nekončí na `SubTest`. S koncovkou `SubTest` som pomenovával pomocné testy, ktoré nemajú byť spúšťané samostatne.

10.1.3 Vlastnosti testovacieho zoznamu

Pri testoch, ktoré sú predpokladom pre iné testy som nastavil vlastnosť „StopTestListOnFailure“, ktorá tak ako už jej názov napovedá, ukončí vykonávanie zoznamu testov v prípade, že daný test zlyhá. Tým sa predíde zlyhaniu testov, ktorých predpoklady by neboli splnené a teda aj nesprávnym výsledkom.

10.2 Vytvorené testovacie zoznamy

Vytvoril som celkovo 9 testovacích zoznamov. Každý z nich popíšem bližšie.

1. **Testovací zoznam pre kategóriu**
Ako prvý som vytvoril zoznam testov pre kategóriu. Nachádzajú sa tu všetky testy testovacích scenárov pre vytvorenie, odstránenie a vytvorenie duplicity kategórie. Tento testovací zoznam je nezávislý na tom, či potrebná kategória, s ktorou pracuje pred jeho spustením existuje alebo neexistuje. Na začiatku testovacieho zoznamu je test, ktorý v prípade, že kategória s názvom získaným z testovacích dát neexistuje, tak ju vytvorí, inak ju vytvárať nebude. Predíde sa tak prípadným zlyhaniam testovacieho zoznamu a potrebe spúšťať osobitný testovací zoznam v prípade, že kategória neexistuje.

2. **Testovací zoznam pre podkategóriu**
V tomto testovacom zozname sa nachádzajú testy patriace testovacím scenárom týkajúcich sa podkategórie.

3. **Testovací zoznam pre vytvorenie produktu**
Prvým z testovacích zoznamov pre produkty je testovací zoznam obsahujúci testy pre vytvorenie produktu, overenie jeho vlastností na frontende a backende.

4. **Testovací zoznam pre odstránenie produktu**
Ďalší testovací zoznam pre produkty obsahuje testy na odstránenie produktu.

5. **Testovací zoznam pre cenu produktu a zľavy**
Tento testovací zoznam obsahuje všetky testy súvisiace s cenou produktu a zľavami. Tieto testy sú zlúčené do jedného testovacieho zoznamu preto, že ak sa vyskytne chyba v cene, tak stačí spustiť jeden zoznam testov pre overenie celkovej funkčnosti ovplyvňujúcej cenu produktu. Patria tu testy pre nastavenie ceny produktu, množstevnej zľavy produktu, zoznamy cien produktu a objemovej zľavy pre všetky produkty.

6. **Testovací zoznam externé odkazy produktu**
V tomto testovacom zozname sú všetky testy súvisiace s externými odkazmi a videami produktu.

7. **Testovací zoznam pre súvisace produkty**
Testovací zoznam pre súvisiace produkty obsahuje testy pre pridanie nových súvisiacich produktov a odstránenie všetkých existujúcich súvisiacich produktov.

8. **Testovací zoznam pre príznaky produktu**
Tento zoznam obsahuje testy pre príznaky produktu, ako sú testy pre nastavenie príznakov, ich overenie a odstránenie.

9. **Testovací zoznam pre varianty produktu**
Do tohto testovacieho zoznamu patria testy pre vytvorenie variant produktu zo šablóny, overenie vytvorených variant a zrušenie používania variant.

Na obrázku 8 vidíme, ako vyzerajú vytvorené testovacie zoznamy v Test Studiu. Počet testov jednotlivých testovacích zoznamov neodpovedá počtu testovacích prípadov, pretože niektoré testy, kvôli už spomínaným nadväznostiam, zahrňujú viac testovacích prípadov.

TYPE	TEST LIST	DATE	OWNER	TESTS
	CategoryAddRemove	10/6/2016 12:25:59 PM		3
	ProductAdd	2/8/2017 9:48:36 AM		4
	ProductExternalLinks	2/22/2017 4:14:29 PM		2
	ProductPriceAndDiscounts	2/22/2017 9:46:06 AM		8
	ProductRelatives	2/22/2017 4:15:21 PM		2
	ProductRemove	2/22/2017 9:47:09 AM		1
	ProductTags	2/22/2017 4:16:21 PM		2
	ProductVariants	2/22/2017 4:17:23 PM		2
	SubcategoryAdd	10/11/2016 8:38:31 AM		2

Obrázok 8: Ukážka testovacích zoznamov v Test Studiu

11 Odporúčania pre písanie zdrojového kódu vývojármi

V tejto kapitole popíšem odporúčania pre písanie zdrojového kódu vývojármi. Tieto odporúčania súvisia s chybami s ktorými som sa stretol a tiež s problémami uvádzanými v kapitole 9.3, ktoré sa pri automatizovanom testovaní môžu vyskytnúť. Dodržiavanie týchto odporúčaní v značnej miere zjednoduší tvorbu automatizovaných testov. K odporúčaniam uvádzam aj ako sa problémy pri ich nedodržiavaní riešia v testovacích nástrojoch, s ktorými mám osobné skúsenosti (Test Studio, Selenium).

11.1 HTML5 vstupy

Jedným z odporúčaní pre písanie kódu vývojármi ovplyvňujúce tvorbu automatizovaných testov je používanie štandardných HTML5 vstupov. Prácu s nimi má Test Studio vstavanú. Pri použití štandardného vstupu pre výber zo zoznamu stačí v kóde priradiť položku, ktorú chceme vybrať podľa jej textu alebo hodnoty. Nemusíme tak na vstup klikať pre otvorenie výberu a ani klikať na položku v zozname pre jej výber.

V Selenium sa v mnohých ohľadoch pracuje rovnako ako v Test Studiu. Pre neštandardné vstupy sa používajú funkcie `SendKeys()` pre vstup z klávesnice a pre kliknutie myši existuje niekoľko metód, ktoré sa správajú podobne ako metódy v Test Studiu.

11.2 Používanie identifikátorov

Identifikátory uľahčujú prácu nie len pri tvorbe automatizovaných testov, ale aj s JavaScriptom a ďalšími technológiami, ktoré pracujú priamo s HTML. Ideálne je, ak sa identifikátor (atribút `id`) nachádza pri čo najväčšom počte HTML elementov, pokiaľ možno aspoň pri každom elemente, s ktorým budú používatelia priamo interagovať, keďže sú tieto elementy testované najčastejšie. Elementy s identifikátorom sa potom dajú jednoducho vyhľadať. Ak má element vlastný identifikátor, tak je v DOM ľahko vyhľadateľný, pretože sa dá nájsť už zadaním samostatného identifikátora napr. pomocou XPath a nie je potreba zadávať ďalšie parametre vyhľadávania.

V prípade, že toto odporúčanie nie je dodržané sa v testovacích nástrojoch používa riešenie pomocou XPath dotazu spomínaného v kapitole 9.3.3.

11.3 Unikátnosť identifikátorov

Identifikátor HTML elementu by mal byť na každej stránke unikátny. Niekoľkokrát som však narazil na problém, že jeden identifikátor sa na stránke nachádzal viackrát. Existujú rozšírenia do prehliadačov, ktoré takéto nedostatky odhalujú. Jedným takým rozšírením je Dup-ID pre Google Chrome.

Problém nastáva keď spustíme test, ktorý sa snaží nájsť element podľa identifikátoru predpokladajúc, že sa tam nachádza takýto element práve jeden. V takom prípade test zlyhá, pretože hľadáme jeden konkrétny element, ale dotaz vráti zoznam elementov. Preto som v XPath dotazoch vždy vyhľadával prvý nájdený prvok. Namiesto dotazu `//input[@id='BasicSection_Name']`, ktorý nám vráti zoznam

vstupov s id rovným `BasicSection\_Name` som tak použil dotaz `(//input[@id='BasicSection_Name'])[1]`, ktorý vráti prvý element z DOM s danou podmienkou.

Keďže tento problém sa vzťahuje na XPath a nie na testovacie nástroje, pre Selenium ako aj pre ďalšie testovacie nástroje používajúce XPath pre vyhľadávanie HTML elementov je riešenie rovnaké.

11.4 Preklepy v hodnotách atribútov HTML elementov

Párkrát som narazil na menšie preklepy, ako napríklad medzera alebo tabulátor v hodnote atribútu class v HTML elemente `<div class="modal-dialog ">`. Nie je to žiadna závažná chyba, no aj tak som na to vývojárov upozornil. Menším problémom však je, že XPath hodnotu class nečíta ako jednotlivé triedy oddelené medzerami, ale ako jeden reťazec znakov. To znamená, že ak budeme vyhľadávať hodnotu class rovnú hodnote `modal-dialog`, tak nám tento element nevráti.

Riešenie je jednoduché a pre Test Studio aj Selenium rovnaké, keďže obidve umožňujú pre vyhľadávanie elementov používať XPath. Pre vyhľadanie elementu použijeme funkciu `contains` alebo `starts-with`. Funkcia `contains` vracia `pravda` alebo `nepravda` v závislosti od toho, či reťazec obsahuje daný podreťazec. Funkcia `starts-with` podobne vracia `pravda`, ak reťazec začína daným podreťazcom, inak vráti `nepravda`. Pre prípad, že sa medzera vyskytne pred názvom triedy je lepšie použiť `contains`. Výsledný XPath dotaz bude vyzerať nasledovne: `//div[contains(@class, 'modal-dialog')]`.

12 Integrovanie automatizovaných testov do procesu zostavenia

Aktuálne sa firma z najväčšej časti spolieha na manuálne testovanie. Jednotkové testy vytvárané programátormi aktuálne nepokrývajú dostatočne veľkú časť zdrojového kódu aplikácie. Snahou je toto pokrytie zvýšiť a zároveň do procesu zaradiť automatizované testovanie, ktoré má z čo najväčšej časti nahradiť časovo náročné manuálne testovanie.

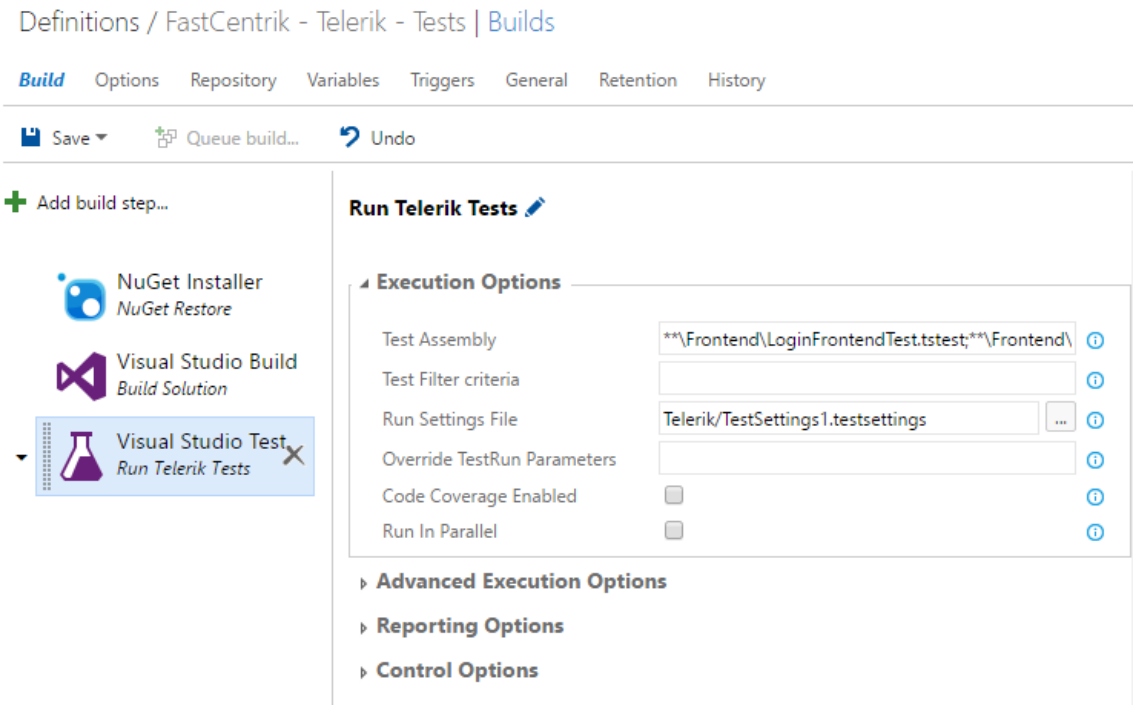
12.1 Inštalácia a konfigurácia

Automatizované testy spúšťame z počítača s nainštalovaným Test Studiom. Pre zjednodušenie ich spúšťania je ešte potrebné ich integrovať do zostavenia. Zostavenie pripravuje aplikáciu na nasadenie, môže zahŕňať procesy ako predspracovanie, kompilovanie, testovanie, vytvorenie balíčku a pod. Zostavenie vykonávame na vzdialenom počítači, a tak zbytočne nezaťažujeme počítač, na ktorom práve pracujeme. Zjednoduší a zefektívni sa tým používanie testov.

Pre zapojenie automatizovaných testov do zostavenia v TFS som musel na vzdialenom počítači nastaviť tzv. agenta. **Agent** je softvér, pomocou ktorého TFS dokáže vykonávať zostavenie alebo nasadenie. Stiahneme si ho z URL `<adresa-tfs-servera>/tfs/_admin/_AgentPool` pomocou odkazu „Download agent“ a nainštalujeme. Agenta som spustil v interaktívnom móde namiesto prednastaveného servisného módu. Je to potrebné preto, aby agent mohol interagovať s počítačom na ktorom je spustený, pretože testy interagujú s webovým prehliadačom. Na tomto vzdialenom počítači musí byť nainštalované Visual Studio a tiež Test Studio, aby spúšťané testy mohli pristupovať k jeho knižniciam.

V TFS som do zoznamu definícií zostavení vytvoril nové zostavenie pre spúšťanie testov. Sú tu kroky, ktoré sa budú vykonávať postupne: inštalácia NuGet balíčkov, Visual Studio Build a Visual Studio Test. Prvé dva kroky pripravia testy na spustenie – nainštalujú sa NuGet balíčky a vykoná sa zostavenie projektu. V treťom kroku, ktorý nás zaujíma najviac sa spustia testy. V nastaveniach tohto kroku som do kolónky „Test Assembly“ zadal cestu k testom, ktoré majú byť spustené. Pomocou `:**\\obj\\**` som určil, že sa majú ignorovať súbory v priečinku `obj`. Nastavenia testov som agentovi predal pomocou súboru s koncovkou `.testsettings`. Cesta k súboru sa zadáva do kolónky „Run Setting File“. V súbore sa nachádzajú nastavenia akými sú základná URL používaná testami, na akom prehliadači majú byť spustené a pod. Nastavenia môžeme vidieť v príklade na obrázku 9.

V záložke „Repository“ som zadal cestu k projektu v rámci repozitára. Máme možnosť pridať aj viac ciest k adresárom v prípade, že to zostavenie vyžaduje.



Obrázok 9: Ukážka nastavení kroku definície zostavenia v TFS

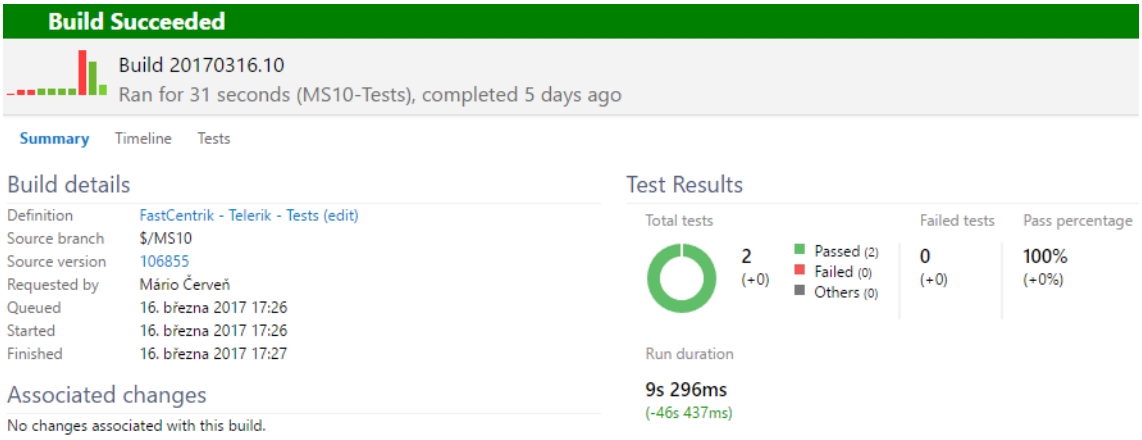
12.2 Spustenie zostavenia

Zostavenie pridáme do fronty tlačidlom „Queue Build“. V prípade, že je fronta prázdna bude spustený ihneď, inak čaká na rad. Beh zostavenia môžeme sledovať pomocou záznamov zobrazovaných v textovom režime podobne ako v konzolovej aplikácii po spustení testov v Test Studiu. Zobrazuje sa tu priebeh testov a úspešnosť jednotlivých krokov.

Po ukončení behu zostavenia si môžeme zobraziť jeho výsledky. V zhrnutí výsledkov (záložka „Summary“, ktoré sú na obrázku 10 je názov zostavenia, či uspel alebo zlyhal, kto ho zahájil, kedy začal a ďalšie detaily. Sú tu aj výsledky testov, v ktorých sú početnosti spustených testov, z toho počet úspešných a neúspešných testov a iné.

V záložke „Timeline“ sa nachádzajú jednotlivé kroky zostavenia. Pri každom sa zobrazuje či prebehol úspešne alebo neúspešne a dĺžka jeho behu.

Záložka „Tests“ zobrazuje podrobnejšie výsledky testov ako sú v zhrnutí výsledkov. Naviac je tu možnosť priamo nahlásiť odhalené chyby.

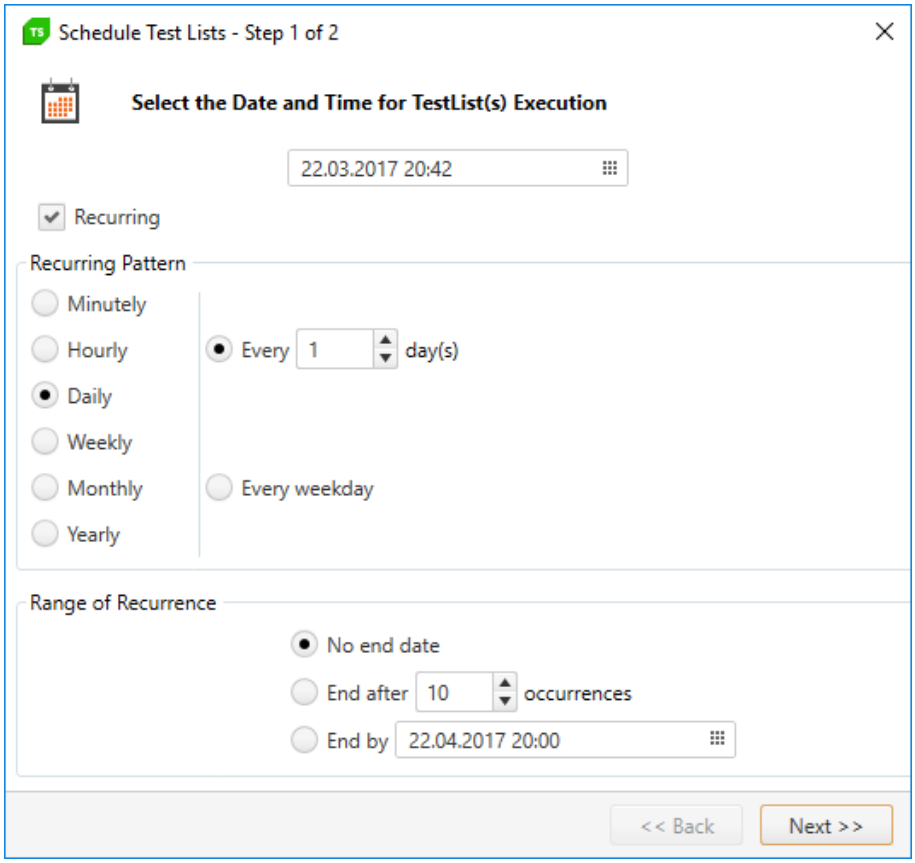


Obrázok 10: Ukážka výsledku zostavenia v TFS

12.3 Naplánovanie spúšťania testovacích zoznamov

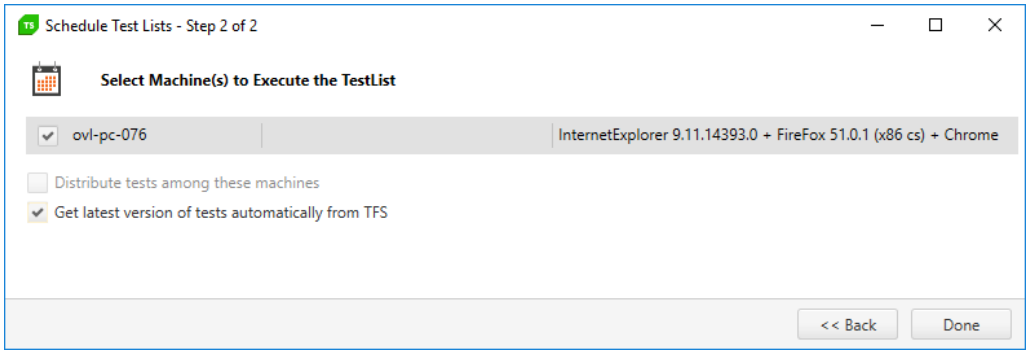
Podľa oficiálnej dokumentácie Test Studia sa pre spúšťanie testovacích zoznamov Test Studia pomocou TFS zostavenia používajú `.vsmdi` súbory, v ktorých sa špecifikuje, ktoré testovacie zoznamy sa majú spustiť. Avšak tieto súbory TFS a Visual Studio od verzie 2013 nepodporujú (ja používam Visual Studio 2015), a preto je najlepšie použiť samotné Test Studio.

Test Studio umožňuje testovacie zoznamy spúšťať na vzdialenom počítači a ich spúšťanie plánovať. V záložke „Test Lists“ som vybral testovací zoznam, ktorý chcem naplánovať a klikol som na tlačidlo „Schedule TestList“. Zobrazila sa ponuka s výberom ako často sa má testovací zoznam spúšťať (obrázok 11). Vybral som opakovanie každý deň, bez konečného dátumu. Vytvorený plán môžeme neskôr zrušiť alebo upraviť.



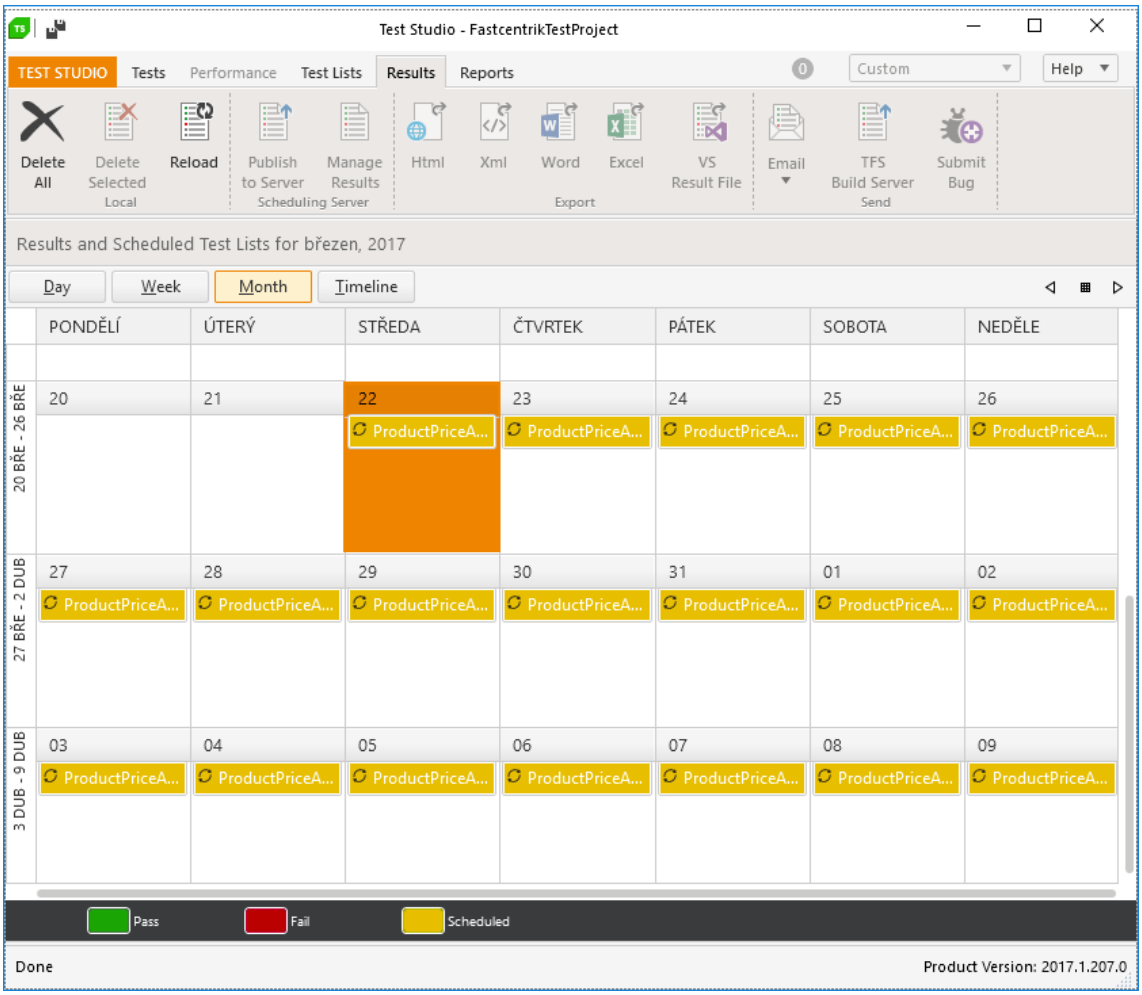
Obrázok 11: Plánovanie testovacieho zoznamu - 1. krok

V ďalšom kroku (obrázok 12) som vybral počítač, na ktorom sa budú testovacie zoznamy spúšťať. Pri tomto počítači sa zobrazujú jeho nainštalované prehliadače. V tomto kroku som ešte nastavil, aby sa pred spustením automaticky získala najnovšia verzia testov z TFS.



Obrázok 12: Plánovanie testovacieho zoznamu - 2. krok

Plán a výsledky testov môžeme vidieť v prehľadnom kalendári (obrázok 13). Výsledky testov sa dajú exportovať do rôznych typov súborov, a tiež odoslať na TFS Build Server alebo mailom. Nájdené chyby máme možnosť rovno nahlásiť.



Obrázok 13: Kalendár plánovania

13 Testovanie verzií webovej aplikácie

Automatizované aj manuálne testy som použil na testovanie piatich verzií aplikácie usporiadané od najstaršej po najnovšiu. Pre zjednodušenie ich nazvime V1 až V5.

Testovanie jednej verzie obidvoma typmi testov spolu s opravami automatizovaných testov a vyhodnocovaním správnosti výsledkov testov trvalo v závislosti od konkrétnej verzie 3 až 7 hodín. Testy som spúšťal nad rôznymi dátami. Testovanie piatich verzií by bolo z časového hľadiska problematické, pretože trvá 2 týždne, kým vyjde nová verzia. Celkovo by táto úloha preto musela prebiehať až počas 10 týždňov. Použil som preto 2 staršie verzie a zredukoval som túto dobu na 6 týždňov.

Verzie V1 a V2 časovo predchádzali verziu V3, na ktorej som testy tvoril. Verzie V4 a V5 som testoval až po ich vzniku a nasadení. Spoločnosť si ukladá zálohy verzií, a tak stačilo staršie verzie pre účel testovania nasadiť. Pracoval som tak s testami staršími aj novšími ako bola verzia na ktorej som ich tvoril.

Testovanie každej verzie malo nasledujúce fázy:

1. vykonávanie testov,
2. spracovanie výsledkov,
3. opravy a optimalizácia.

V prípade odhalenia nejakého nedostatku testov počas ich vykonávania, som tieto nedostatky napravil v tzv. fáze opráv a optimalizácie. Nedostatky mohli byť nájdené v testovacích prípadoch ako aj v testovacích skriptoch.

13.1 Účel testovania verzií

Vďaka použitiu až piatich verzií som mohol porovnať ich kvalitu z hľadiska funkcionality, čo bolo aj hlavným cieľom testovania verzií. Ďalším učelom bolo použitie automatizovaných testov na viacerých verziách aplikácie, ktorých odlišnosti odhalili nedostatky niekoľkých testov a pomohli tak k ich optimalizácii. Opravy testov pre všetky verzie pozitívne ovplyvnia aj testovanie do budúcna. V aplikácii som tiež našiel niekoľko chýb, ktoré som poslal na opravu. Navyše použitím manuálnych aj automatizovaných testov som mohol porovnať ich efektívnosť.

Pre každú testovanú verziu som zostavil tabuľku úspešnosti testov. Z nej som potom zisťoval, či boli nájdené vady v ďalšej verzii opravené, či sa nejaké nové vady vyskytli, z testovaných oblastí aplikácie som určil najporuchovejšiu a pod. Získané výsledky som poslal manažmentu na ďalšie spracovanie a vývojárom pre vykonanie opráv vád.

V prípade, že manuálny test prebehol neúspešne, tak bol rovnako neúspešný aj k nemu priradený automatizovaný test. Avšak nastali prípady, kedy manuálny test prebehol úspešne a automatizovaný kvôli odlišnostiam verzií neúspešne. Vtedy bolo nutné vykonať potrebné úpravy, aby bol aplikovateľný aj na danú verziu.

13.2 Testovanie verzie V1

Ako prvú som testoval verziu V1. Táto verzia bola v dobe testovania o 2 verzie staršia ako aktuálna, a preto bolo predpokladom, že pre ňu bude potrebný väčší počet úprav testov.

13.2.1 Popis výsledkov a úprav testov

Pri testovaní tejto verzie som narazil na chybu v automatizovanom teste s názvom Pridanie kategórie, ktorý zlyhal v prípade, že nebol vyplnený nepovinný vstup. V zdrojovom kóde som pre zadávanie textu z klávesnice používal funkciu, ktorá pre prázdny reťazec vyhodí výnimku. K oprave stačilo overiť prázdny reťazec, viď výpis 10. Použitím rôznych testovacích dát som odhalil niekoľko podobných chýb v testoch.

```
string text = (string) Data["alternativeName"];

if (text != string.Empty)
{
   ActiveBrowser.Find.ById("BasicSection_AlternativeName").Focus();
   Manager.Desktop.KeyBoard.TypeText(text, 20, 20, true);
}
```

Výpis 10: Oprava testu Pridanie kategórie

Ďalej som zistil, že test Pridanie súvisiacich produktov nesprávne pokračoval vo vykonávaní v prípade, že nenašiel hľadaný súvisiaci produkt. Túto chybu som opravil, aby v takom prípade zlyhal a ďalej nepokračoval.

V niekoľkých ďalších testov stačilo zmeniť vyhľadávanie elementov. Medzi tieto testy patrili: Vytvorenie variant produktu zo šablóny, Odstránenie všetkých súvisiacich produktov a Nastavenie ceny produktu, ktoré pracovali v záložkách detailu produktu. Tieto záložky neobsahovali atribút `data-id` pomocou ktorého som sa na ne odkazoval, a preto som odkazovanie na ne zovšeobecnil.

Jediný test, ktorý pre verziu V1 zlyhal pre manuálny aj automatizovaný test je Odstránenie produktu. Produkt sa síce odstránil, ale jeho URL adresa stále existovala v databáze, a preto nebolo možné vytvoriť nový produkt s rovnakou URL adresou.

13.2.2 Výsledná úspešnosť testov

Početnosti úspešných a neúspešných testov som zapísal do tabuľky, ktorú vidíme na obrázku 14. Zameriame sa na neúspešné testy, pretože tie nás zaujímajú najviac.

Neúspešných automatizovaných testov bolo najskôr 6, po úpravách sa tento počet znížil na 1. Upravil som tak celkovo 5 automatizovaných testov. Spomínaný predpoklad, že počet neúspešných automatizovaných testov pre túto verziu bude vysoký bol splnený. Zároveň však bolo aj opravených a zovšeobecnených veľký počet testov, a tak sa predpokladá, že počet opráv pre ďalšie verzie bude nižší.

Z celkového počtu 49 testov bol neúspešný 1 manuálny, ktorý bol zároveň neúspešný pre automatizovaný test. Tento test je už spomínaný test pre odstránenie produktu, patriaci do skupiny Základná funkcionalita produktu. Táto funkcionalita bola správne určená ako chybová. Z testovanej funkcionality verzie V1 teda bola najporuchovejšia oblasť základnej funkcionality produktu s jednou odhalenou chybou.

Po úpravách automatizovaných testov sa výsledky oboch typov testov zhodli na tom, že z 49 testov bol 1 neúspešný, čo vo výsledku znamená chybovosť verzie 2,0%.

Verzia V1	Počet testovacích scenárov	Manuálne testy		Automatizované testy (pred úpravami)		Automatizované testy (po úpravách)	
Skupina test. scenárov		Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných
Prihlásenie	2	2	0	2	0	2	0
Kategórie	10	10	0	9	1	10	0
Zákl. funkcionalita produktu	8	7	1	7	1	7	1
Cena produktu	5	5	0	4	1	5	0
Zľavy produktu	3	3	0	3	0	3	0
Výrobci produktu	1	1	0	1	0	1	0
Súvisiace produkty	3	3	0	1	2	3	0
Varianty produktu	3	3	0	2	1	3	0
Externé odkazy produktu	3	3	0	3	0	3	0
Príznaky produktu	3	3	0	3	0	3	0
Zľavy	6	6	0	6	0	6	0
Ostatné	2	2	0	2	0	2	0
Spolu	49	48	1	43	6	48	1

Obrázok 14: Tabuľka úspešnosti testov pre verziu V1

13.3 Testovanie verzie V2

Verziu V2 som testoval v slovenskej lokalizácii. Testy boli najskôr tvorené pre českú lokalizáciu, no snažil som sa ich od začiatku robiť čo najmenej závislé na akýchkoľvek textoch a lokalizácii. Napriek tomu sa však môžu vyskytnúť problémy, kde nebude HTML element nájdený kvôli prekladu. Tieto problémy som sa snažil nájsť a opraviť. Okrem chýb v tejto verzii som tým mohol odhaliť a odstrániť závislosť testov na lokalizácii a tým umožniť ich aplikovateľnosť pre všetky existujúce lokalizácie, pretože predpokladom je, že ak sú aplikovateľné na 2 lokalizácie, budú aplikovateľné aj na ďalšie. Aktuálne je aplikácia lokalizovaná okrem češtiny a slovenčiny aj do poľštiny, angličtiny a nemčiny. Slovenskú lokalizáciu som testoval preto, že je spolu s českou najviac využívaná a zároveň aj preto, že jej najviac rozumiem.

Nezávislé na lokalizácii sú testy aj testovacie dáta. S desatinnými číslami som napríklad pracoval tak, že nezáleží, či použijeme číslo s desatinou čiarkou alebo bodkou.

13.3.1 Popis výsledkov a úprav testov

V tejto verzii bola vytvorená nová grafická šablóna. Túto šablónu som testoval a odhalil som pri nej 2 chyby. Pri jej použití zlyhali manuálne aj automatizované testy Overenie súvisiacich produktov a Overenie variant produktu, a to preto, že sa na frontende pri produkte nezobrazovali súvisiace produkty ani varianty.

Pre obidva typy testov zlyhalo aj Overenie ceny produktu v prípade, že som v testovacích dátach nastavil dopravu zdarma. Podľa špecifikácie by sa mal k produktu automaticky pridať príznak dopravy zdarma, no nestalo sa tak.

Kvôli odlišnej lokalizácii zlyhali 2 automatizované testy. Prvým z nich bol test Prihlásenie do frontendu, ktorý zlyhal pri overovaní, či existuje odkaz s URL `\prihlaseni`. Ten je v slovenskej lokalizácii `\prihlasenie`. Druhý zlyhal, pretože nebolo nájdené tlačidlo pre návrat na predchádzajúcu stránku v teste Pridanie kategórie. Oprava testov pre lokalizáciu bola jednoduchá, stačilo zovšeobecniť vyhľadávanie HTML elementov tak, aby neboli závislé na lokalizácii.

13.3.2 Výsledná úspešnosť testov

Z obrázku 15 vidíme, že z 49 testov boli neúspešné 3 manuálne a 5 automatizovaných. Po úpravách sa počet neúspešných automatizovaných testov znížil na 3. Upravil som 2 automatizované testy, ktoré boli závislé na lokalizácii. Chyba základnej funkcionality produktu z predchádzajúcej verzie bola opravená, ale vyskytli sa až 3 nové chyby.

Všimnime si, že počet opráv testov sa znížil z 5 vo verzii V1 na 2 vo V2. Vykonané opravy vo V1 zlepšili aplikovateľnosť testov aj pre ďalšie verzie.

Z 49 testov boli neúspešné 3 čo znamená, že chybovosť verzie V2 je 6,1%.

Verzia V2	Počet testovacích scenárov	Manuálne testy		Automatizované testy (pred úpravami)		Automatizované testy (po úpravách)	
Skupina test. scenárov		Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných
Prihlásenie	2	2	0	1	1	2	0
Kategórie	10	10	0	9	1	10	0
Zákl. funkcionalita produktu	8	8	0	8	0	8	0
Cena produktu	5	4	1	4	1	4	1
Zľavy produktu	3	3	0	3	0	3	0
Výrobci produktu	1	1	0	1	0	1	0
Súvisiace produkty	3	2	1	2	1	2	1
Varianty produktu	3	2	1	2	1	2	1
Externé odkazy produktu	3	3	0	3	0	3	0
Príznaky produktu	3	3	0	3	0	3	0
Zľavy	6	6	0	6	0	6	0
Ostatné	2	2	0	2	0	2	0
Spolu	49	46	3	44	5	46	3

Obrázok 15: Tabuľka úspešnosti testov pre verziu V2

13.4 Testovanie verzie V3

Ako ďalšiu som testoval verziu V3. Na tejto verzii som vytváral väčšinu testov, a preto je očakávané, že počet automatizovaných testov, ktoré bude treba opraviť bude nízky.

13.4.1 Popis výsledkov a úprav testov

V tejto verzii bol neúspešný len jeden manuálny test, ktorým je test s názvom Skryť kategóriu. Po spustení automatizovaného testu sa skrytá kategória síce v zozname kategórií nezobrazovala, ale test odhalil, že keď sa dostaneme na jej URL adresu tak sa nám aj tak zobrazí. Požiadavkom je, aby sa v takomto prípade zobrazila chyba 404.

13.4.2 Výsledná úspešnosť testov

Z tabuľky na obrázku 16 môžeme vidieť, že neúspešný bol len jeden manuálny test, ktorým bol spomínaný test Skryť kategóriu. Keďže som na tejto verzii vytváral väčšinu automatizovaných testov, žiadne z nich som nemusel upravovať. Počet neúspešných automatizovaných a manuálnych testov sa v tomto prípade nelíšia.

Jeden neúspešný test vo V3 z celkového počtu 49 dáva chybovosť 2,0%.

Verzia V3	Počet testovacích scenárov	Manuálne testy		Automatizované testy (pred úpravami)		Automatizované testy (po úpravách)	
Skupina test. scenárov		Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných
Prihlásenie	2	2	0	2	0	2	0
Kategórie	10	9	1	9	1	9	1
Zákl. funkcionalita produktu	8	8	0	8	0	8	0
Cena produktu	5	5	0	5	0	5	0
Zľavy produktu	3	3	0	3	0	3	0
Výrobci produktu	1	1	0	1	0	1	0
Súvisiace produkty	3	3	0	3	0	3	0
Varianty produktu	3	3	0	3	0	3	0
Externé odkazy produktu	3	3	0	3	0	3	0
Príznaky produktu	3	3	0	3	0	3	0
Zľavy	6	6	0	6	0	6	0
Ostatné	2	2	0	2	0	2	0
Spolu	49	48	1	48	1	48	1

Obrázok 16: Tabuľka úspešnosti testov pre verziu V3

13.5 Testovanie verzie V4

Testovaná verzia V4 vznikla až po vytvorení testov. Automatizované testy už boli pri testovaní tejto verzie stabilné, a preto sa predpokladá, že nebude nutné upravovať väčšie množstvo automatizovaných testov.

13.5.1 Popis výsledkov a úprav testov

Bola odhalená chyba pri teste Vyprázdnenie košíka po jednotlivých položkách. Pri pokuse odstrániť položku z košíka sa pri istých nastaveniach zobrazila chybová hláška, že sa nepodarilo nájsť položku. Jediným riešením bolo odstrániť cookies prehliadača.

Ďalšou odhalenou chybou bolo nesprávne zobrazovanie názvu podkategórie v odkaze na ňu.

Z automatizovaných testov naviac zlyhal test Overenie obrázkov produktu. Ten na frontende overoval hodnotu atribútu `alt` obrázkov produktu. Tento atribút špecifikuje text, ktorý sa zobrazí v prípade, že sa nepodarí zobraziť obrázok. V predchádzajúcich verziách aplikácie bola hodnota rovná názvu obrázku. V tejto verzii sa hodnota zmenila na cestu k obrázku. Táto cesta môže byť na rôznych projektoch platformy odlišná, a preto som si z nej jednoducho získal názov obrázku a overil ho s očakávaným názvom, ktorý sa získava v administrátorskom rozhraní, viď výpis 11.

```
HtmlImage secondaryImage = ActiveBrowser.Find.ByExpression<HtmlImage>(
    secondaryImageExpr + "[" + i + "]");
string secondaryImageName = secondaryImage.GetValue<string>("alt");
secondaryImageName = secondaryImageName.Substring(secondaryImageName.
    LastIndexOf('/') + 1);
```

Výpis 11: Oprava testu Overenie obrázkov produktu

13.5.2 Výsledná úspešnosť testov

Výsledná tabuľka z obrázku 17 hovorí, že pri manuálnom testovaní boli odhalené 2 chybné funkcionality. Sú nimi Zobrazenie podkategórie a Vyprázdnenie košíka po jednotlivých položkách.

Z automatizovaných testov bolo potrebné verzii prispôsobiť len jeden test – Overenie obrázkov produktu. Tento nízky počet potrebných zmien potvrdil, že automatizované testy sú dostatočne optimalizované pre použitie aj pre ďalšie budúce verzie.

Výsledný počet úspešných testov je 47 a počet neúspešných je 2, čo činí výslednú chybovosť verzie 4,1%.

Verzia V4	Počet testovacích scenárov	Manuálne testy		Automatizované testy (pred úpravami)		Automatizované testy (po úpravách)	
Skupina test. scenárov		Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných
Prihlásenie	2	2	0	2	0	2	0
Kategórie	10	9	1	9	1	9	1
Zákl. funkcionalita produktu	8	8	0	7	1	8	0
Cena produktu	5	5	0	5	0	5	0
Zľavy produktu	3	3	0	3	0	3	0
Výrobci produktu	1	1	0	1	0	1	0
Súvisiace produkty	3	3	0	3	0	3	0
Varianty produktu	3	3	0	3	0	3	0
Externé odkazy produktu	3	3	0	3	0	3	0
Príznaky produktu	3	3	0	3	0	3	0
Zľavy	6	6	0	6	0	6	0
Ostatné	2	1	1	1	1	1	1
Spolu	49	47	2	46	3	47	2

Obrázok 17: Tabuľka úspešnosti testov pre verziu V4

13.6 Testovanie verzie V5

Verziu V5 som testoval po jej predchodcovi V4. Rovnako ako pri V4 sa očakáva, že nebude nutné upravovať väčšie množstvo automatizovaných testov. Očakávam tiež, že tak ako tomu bolo v predchádzajúcich verziách, budú predošlé odhalené chyby vo V5 opravené.

13.6.1 Popis výsledkov a úprav testov

Test Overenie validácie ceny produktu zo skupiny ceny produktu zlyhal, čím odhalil, že sa vo viackrokovej objednávke nezobrazovala výsledná cena dopravy.

Zlyhal aj test Nastavenie príznakov produktu patriaci do skupiny príznaky produktu. Pri pokuse uložiť vybrané príznaky produktu v administrátorskom rozhraní sa príznaky síce uložili, ale následne bola zobrazená chyba 500 (Internal Server Error).

Výsledky automatizovaných testov sa zhodovali s manuálnymi, neboli vyžadované žiadne úpravy.

13.6.2 Výsledná úspešnosť testov

Z tabuľky na obrázku 18 vidíme, že zlyhali 2 automatizované aj manuálne testy. V obidvoch prípadoch sú to Overenie validácie ceny produktu a Nastavenie príznakov produktu.

Žiadne automatizované testy nebolo treba upravovať, čo je pre nás žiadaný stav, a tak sú pripravené na použitie aj pre ďalšie budúce verzie.

Z 49 testov boli neúspešné 2, čo znamená chybovosť na úrovni 4,1%.

Verzia V5	Počet testovacích scenárov	Manuálne testy		Automatizované testy (pred úpravami)		Automatizované testy (po úpravách)	
Skupina test. scenárov		Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných	Počet úspešných	Počet neúspešných
Prihlásenie	2	2	0	2	0	2	0
Kategórie	10	10	0	10	0	10	0
Zákl. funkcionalita produktu	8	8	0	8	0	8	0
Cena produktu	5	4	1	4	1	4	1
Zľavy produktu	3	3	0	3	0	3	0
Výrobci produktu	1	1	0	1	0	1	0
Súvisiace produkty	3	3	0	3	0	3	0
Varianty produktu	3	3	0	3	0	3	0
Externé odkazy produktu	3	3	0	3	0	3	0
Príznaky produktu	3	2	1	2	1	2	1
Zľavy	6	6	0	6	0	6	0
Ostatné	2	2	0	2	0	2	0
Spolu	49	47	2	47	2	47	2

Obrázok 18: Tabuľka úspešnosti testov pre verziu V5

13.7 Porovnanie kvality verzií

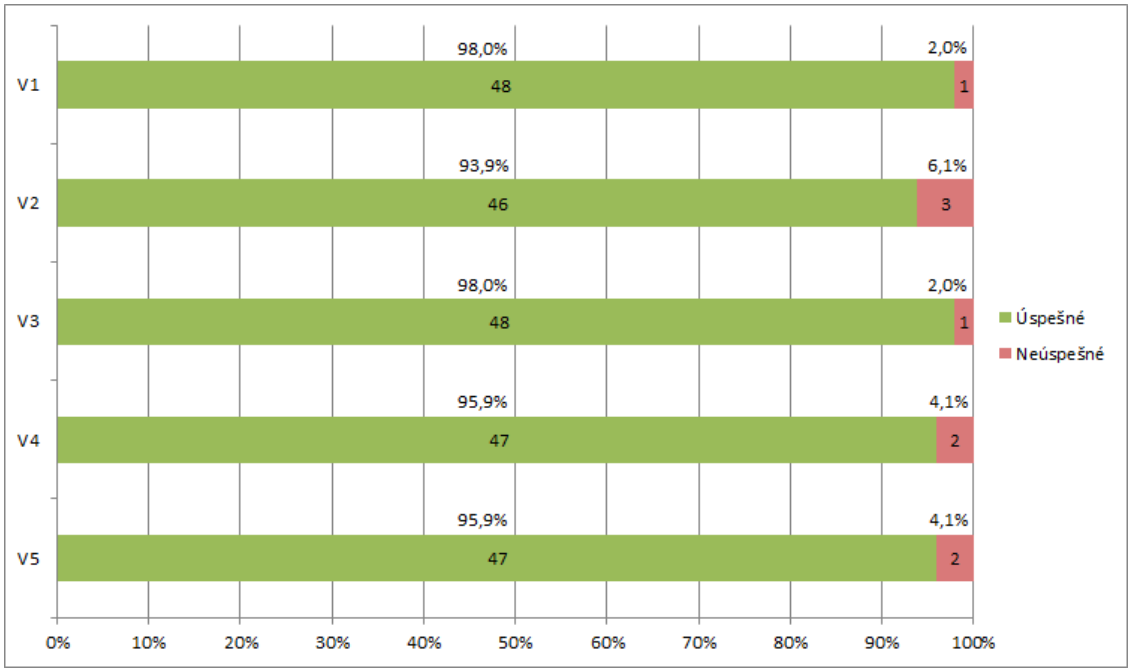
Výsledné početnosti manuálnych a automatizovaných testov som zapísal do tabuľky. Početnosti sú rozdelené pre úspešné a neúspešné testy a ďalej priradené k príslušným verziám aplikácie.

Z grafu obrázka 19 vidíme vývoj počtu chýb testovanej časti naprieč verziami. Tento graf počtu chýb manuálnych testov sa zhoduje s reálnym počtom chýb. Z grafu je zrejmé, že počet chýb je z dlhodobého hľadiska ustálený, pretože počet chýb kolísa len mierne. Zároveň môžeme vidieť, že počet chýb sa ani v jednej z verzií nedostal na hodnotu 0. Pre často sa objavujúce nové chyby by bolo obtiažne sa na takej hodnote z dlhodobého hľadiska udržať. Tiež je z grafu zrejmé, že počet chýb nemá čisto klesajúce tempo, keďže napr. z V1 na V2 sa počet chýb zvýšil.

Dobrou správou je, že žiadna z chýb sa medzi verziami neopakovala, čo znamená, že všetky boli medzičasom opravené. K tomu som prispel aj ja ich odhalením a nahlásením.

Z testovania verzií vieme, že niekoľko chýb sa vždy odhalí až v produkcii po nasadení. Takéto neskoré odhalenie chyby je finančne náročné, riešením je odhalenie chýb pred tým, než sa dostanú do produkcie. Počas implementácie novej funkcionality a tiež pri opravách už existujúcich chýb vznikajú často nové chyby. Práve tento problém rieši regresné testy, a preto mnou vytvorené automatizované testy slúžiace ako funkčné a regresné testy budú mať pre firmu veľký prínos.

Verziou s najväčším počtom chýb je V2 s troma chybami. Z hľadiska funkčnosti ju označíme za najmenej kvalitnú. Naopak za funkčne najkvalitnejšie verzie považujeme V1 a V3, pretože obidve mali najmenší počet chýb (1). Vo V1 a V3 boli tieto chyby odlišné, ale ani jedna z nich nebola kritická.



Obrázok 19: Graf úspešnosti manuálnych testov

Najproblematickejšími oblasťami aplikácie sú oblasti Cena produktu a Varianty produktu. Každá z nich má naprieč všetkými verziami 2 chyby. Najmenej problémových bolo 5 oblastí, v ktorých neboli nájdené žiadne chyby (Prihlásenie, Zľavy produktu, Výrobcovia produktu, Externé odkazy produktu, Zľavy).

Najkritickejšie chyby boli vo V2 (Overenie súvisiacich produktov a Overenie variant produktu) a vo V4 (Vyprázdnenie košíka po jednotlivých položkách).

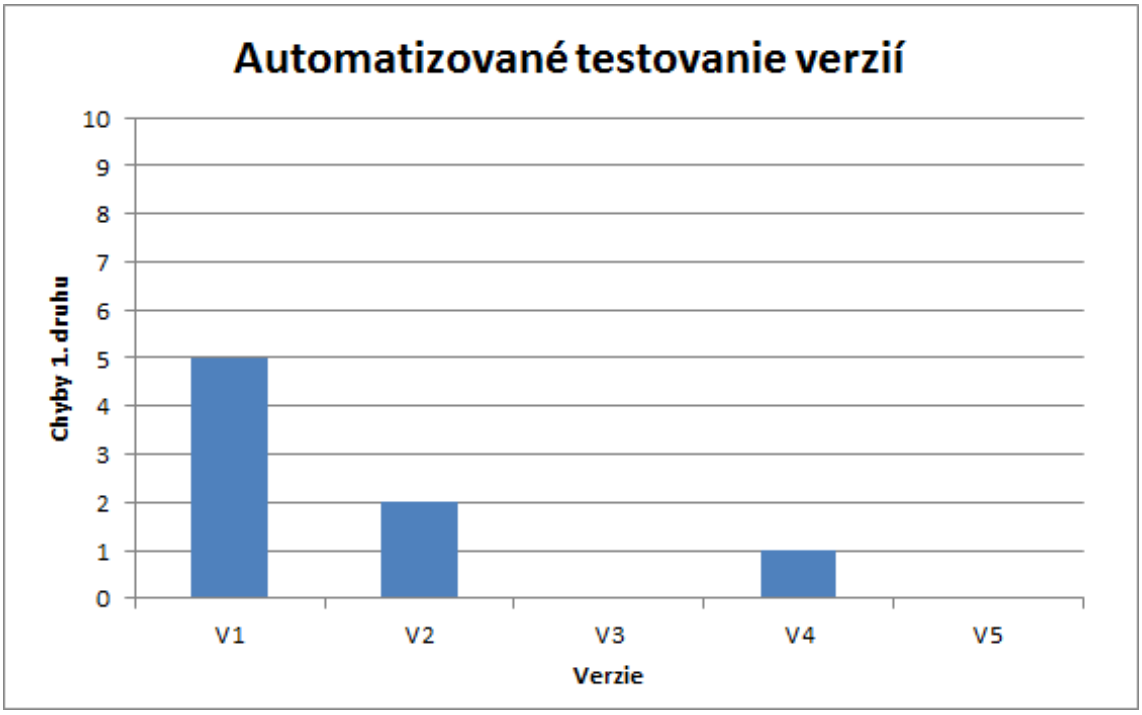
13.8 Zistené chyby v automatizovaných testoch

Pri automatizovaných testoch párkrát došlo k tzv. chybám 1. druhu, nazývaných tiež falošným poplachom – test zlyhal aj napriek tomu, že mal prebehnúť úspešne. Obecne chyba 1. druhu nastala najčastejšie v prípadoch, keď bola vykonaná zmena v UI, ktorá test ovplyvnila. Najčastejšie to bolo z nasledujúcich dôvodov:

- Nebol nájdený HTML element, pretože sa nachádzal na inom mieste na stránke alebo mal odlišné hodnoty atribútov ako tie, na ktoré som sa dotazoval. V takýchto prípadoch som zovšeobecnil XPath dotaz.
- V starších verziách boli odlišné a často aj prázdne dáta. Vďaka tomu som našiel pár chýb v testoch a opravil ich tak, aby boli aplikovateľné na rôzne dáta aplikácie ako aj pre rôzne vstupné testovacie dáta.

- Test spočiatku nemusel byť plne prispôsobený rôznym lokalizáciám a odlišným URL adresám odkazov. V opravách som odstránil akékoľvek závislosti na lokalizácii.

Opravy takýchto chýb v testoch sú väčšinou časovo menej náročné. Po skúsenostiach s automatizovanými testami je pre nás dôležitý jeden poznatok – zakaždým, keď test zlyhá alebo vráti neočakávané výsledky, je potrebné tieto výsledky hlbšie preskúmať.



Obrázok 20: Graf počtu chýb 1. druhu automatizovaných testov

Z grafu na obrázku 20 vidíme, že počet chýb 1. druhu bol pri automatizovaných testoch pri prvom testovaní vysoký. Po každom testovaní som tieto chyby opravil, čím sa z pôvodných 5 chýb 1. druhu vo verzii V1 tento počet zredukoval na 0 pre V3. Vždy sa však v ďalšej verzii môžu vyskytnúť nové, tak ako tomu bolo vo V4. Z dlhodobého hľadiska sa však tento počet postupne znížil.

Okrem chýb 1. druhu existujú aj chyby 2. druhu. Sú to chyby falošného tvrdenia, že funkcia je správna. Napriek tomu, že v skutočnosti dochádza k zlyhaniu, toto zlyhanie nie je ohlásené. Chyby tohto typu môžu byť prejavom nadmernej dôveryhodnosti. Pri automatizovanom testovaní sa z pravidla vyskytujú v menšom počte než chyby 1. druhu.

13.9 Prínos testovania verzií

Testovanie verzií malo okrem už spomínaných cieľov aj nasledujúce prínosy:

- Bola znížená závislosť testov na štruktúre DOM.
- V testoch bolo nájdených a opravených niekoľko chýb.
- Testy viac nie sú závislé na lokalizácii.
- Testy sú stabilnejšie a dôveryhodnejšie aj pre budúce verzie aplikácie.

Na testovanie som použil 2 staršie verzie, ktoré mali odlišné dáta než verzia na ktorej som testy vytváral a v niektorých prípadoch dokonca neboli dáta žiadne. To malo za následok odhalenie niekoľkých nedostatkov v testoch, ktoré na aktuálnej verzii nebolo možné zistiť. Tieto nedostatky som opravil a testy sú tak aplikovateľné na širšiu škálu dát.

Všetky tieto prínosy sú veľkou výhodou pre budúce verzie, keďže sa zvýšila pravdepodobnosť, že automatizované testy na nich budú fungovať bez potrebných úprav.

Z použitia manuálneho aj automatizovaného testovania som dospel k záveru, že pri každej ďalšej verzii je pri prvom spustení automatizovaných testov potreba pre každý neúspešný test zistiť, či sa jedná o chybu aplikácie alebo testu. To môžeme väčšinou spoznať jednoducho už zo samotnej chybovej hlášky a zo snímky obrazovky v čase chyby, ktorú nám Test Studio poskytuje. V prípade, že je chyba na strane testu, musíme túto chybu zanalyzovať a opraviť. Stále však platí, že tento prístup je časovo menej náročný než manuálne testovanie. Ak budú splnené odporúčania pre vývojárov spomínané v kapitole 11, tak sa čas opravy testov zredukuje na minimum a v niektorých prípadoch až úplne odstráni. Z vybraných odporúčaní je v tomto prípade najdôležitejším používanie identifikátorov elementov.

Pri manuálnych testoch takýto problém samozrejme nemáme, vďaka čomu sú menej chybové pri aplikovaní na novšiu verziu.

14 Záver

Hlavným cieľom tejto práce bolo vytvorenie automatizovaných testov v Test Studiu pre dôležité časti aplikácie. Ďalším cieľom bolo pomocou vytvorených automatizovaných a manuálnych testov otestovať verzie aplikácie a porovnať ich kvalitu. Tvoril som testy pre funkčné (testovanie funkcionality) a regresné testovanie UI webovej aplikácie.

Celkovo som vytvoril 49 testovacích scenárov pre najdôležitejšie časti aplikácie a rozdelil som ich na skupiny a podskupiny podľa základnej štruktúry aplikácie. Pre testovacie scenáre som potom vytváral automatizované testy v Test Studiu. Tvorba automatizovaných testov okrem samotného programovania testov obnášala aj ďalšie činnosti, ako príprava testovacích dát a naprogramovanie pomocných metód.

V Test Studiu som testy rozdelil do niekoľkých testovacích zoznamov. V nich sú spúšťané v nastavenom poradí za sebou. Poradie som vyberal tak, aby testy na začiatku zoznamu splnili predpoklady testov po nich nasledujúcich. Tým som dosiahol, že testovacie zoznamy ako jednotlivé celky majú minimum povinných predpokladov alebo dokonca nemajú predpoklady žiadne, vďaka čomu sú prispôsobené aj pre spúšťanie na vzdialenom počítači a bez ľudského zásahu.

Vytvorené automatizované testy som zapojil do životného cyklu produktu. Zabezpečil som jednoduchosť, rýchlosť a bezprostrednosť ich použitia. Vďaka tomu sú firmou pravidelne používané.

Pomocou automatizovaných a manuálnych testov som testoval 5 verzií webovej aplikácie. Tieto verzie sa odlišovali okrem zmien vo funkcionalite a v UI aj v tom, že jedna z nich mala odlišnú lokalizáciu (bola lokalizovaná do slovenčiny, ostatné do češtiny). Kvôli všetkým odlišnostiam medzi verziami som dokázal nájsť a opraviť niekoľko menších nedostatkov automatizovaných testov, eliminovať ich závislosť na lokalizácii, zovšeobecniť ich a zvýšiť ich použiteľnosť. Tým sa pre budúce verzie aplikácie značne znížil potrebný čas na údržbu automatizovaných testov.

Kvalitu jednotlivých verzií aplikácie som porovnal a výsledky som zapísal. Vďaka testovaniu verzií som dospel k záveru, že v prípade splnenia odporúčaní pre vývojárov (hlavne používaním identifikátorov pri HTML elementoch), ktoré spomínam v kapitole 11, sa v závislosti od zmien medzi verziami čas opravy testov zredukuje na úplné minimum alebo opravy nebudú vôbec potrebné.

Diplomová práca mala pre mňa veľký prínos. Získal som množstvo nových vedomostí, naučil som sa vytvárať testovacie prípady a osvojil som si niektoré techniky testovania. Nadobudol som praktické skúsenosti od návrhu, cez tvorbu až po údržbu automatizovaných testov. Prešiel som si celým procesom testovania softvéru. Veľkým prínosom pre mňa je aj to, že som sa naučil pracovať v pokročilom testovacom nástroji, akým je Test Studio.

15 Referencie

[1] PATTON, Ron. *Software Testing*. Indianapolis: Sams Publishing, 2001, 273 s. ISBN 0-672-31983-7.

[2] *Obrázok: Cena opravy chyby.* [online]. [cit. 4. 1. 2017].
Dostupné na
http://www.agilemodeling.com/essays/costOfChange.htm

[3] *Scrum Desk* [online]. [cit. 4. 1. 2017].
Dostupné na
http://scrum.sk/o-agile-a-scrum/scrum-agilny-projektovy-manazment/

[4] *Scrum Alliance* [online]. [cit. 5. 1. 2017].
Dostupné na
https://www.scrumalliance.org/why-scrum

[5] *Obrázok: Vizualizácia Scrumu.* [online]. [cit. 5. 1. 2017].
Dostupné na
https://blogs.msdn.microsoft.com/jmeier/2014/05/29/scrum-at-a-glance-visual/

[6] *NetDirect s.r.o.* [online]. [cit. 8. 1. 2017].
Dostupné na
http://www.netdirect.cz

[7] *The Testing Standards Working Party* [online]. [cit. 8. 1. 2017].
Dostupné na
http://www.testingstandards.co.uk/bs_7925-1_online.htm

[8] *ISTQB Foundation Level and Agile Tester Certification guide* [online]. [cit. 12. 1. 2017].
Dostupné na
http://istqbexamcertification.com/

[9] *Dimenzie kvality* [online]. [cit. 12. 1. 2017].
Dostupné na
http://softwaretestingfundamentals.com/dimensions-of-software-quality/

[10] *Tutorialspoint* [online]. [cit. 13. 1. 2017].
Dostupné na
https://www.tutorialspoint.com/software_testing/software_testing_documentation.htm

[11] *PSI - Testovanie softvéru manuálne* [online]. [cit. 13. 1. 2017].
Dostupné na

http://www2.fiit.stuba.sk/~bielik/courses/
psi-slov/slajdy/psi-manualne-testovanie2014.pdf

[12] *Quality Assurance vs Quality Control* [online]. [cit. 24. 1. 2017].
Dostupné na
http://www.diffen.com/difference/Quality_Assurance_vs
_Quality_Control

[13] *Telerik Test Studio Documentation* [online]. [cit. 8. 2. 2017].
Dostupné na
http://docs.telerik.com/teststudio/

[14] *Learn QTP* [online]. [cit. 8. 2. 2017].
Dostupné na
http://www.learnqtp.com/

[15] *SeleniumHQ* [online]. [cit. 10. 2. 2017].
Dostupné na
http://www.seleniumhq.org/

[16] *FastCentrik* [online]. [cit. 12. 2. 2017].
Dostupné na
https://www.fastcentrik.cz

A Priložené CD

Priložené CD obsahuje:

- thesis – adresár s textom práce vo formáte PDF,
- test_scenarios – adresár obsahujúci dokument s testovacími scenármi a testovacími prípadmi,
- test_results – adresár obsahujúci dokument s výsledkami testovania verzií,
- teststudio_project – adresár projektu vytvoreného v Test Studiu.

B Zoznam testovacích scenárov

V tejto prílohe je výpis vytvorených testovacích scenárov a ich rozdelených do skupín.

B.1 Prihlásenie

Testovacie scenáre: Prihlásenie do frontendu, Prihlásenie do administrátorského rozhrania

B.2 Kategórie

Testovacie scenáre: Pridanie kategórie, Validácia formuláru kategórie, Zobrazenie kategórie, Odstránenie kategórie, Pridanie podkategórie, Zobrazenie podkategórie, Skryť kategóriu, Duplikovanie kategórie, Nastavenie SEO kategórie, Zobrazenie SEO kategórie

B.3 Produkty

- **Základná funkcionalita produktu**
  Testovacie scenáre: Pridanie produktu, Overenie validácie produktu, Vyhľadanie produktu, Nastavenie SEO produktu, Overenie SEO produktu, Odstránenie produktu, Nastavenie obrázkov produktu, Overenie obrázkov produktu
- **Cena produktu**
  Testovacie scenáre: Nastavenie ceny produktu, Overenie validácie ceny produktu, Overenie ceny produktu, Nastavenie a overenie minimálnej predajnej ceny produktu, Nastavenie a overenie obvyklej (katalógovej) ceny produktu
- **Zľavy produktu**
  Testovacie scenáre: Overenie množstevnej zľavy produktu, Overenie validácií množstevnej zľavy produktu, Overenie zľavy a limity zľavy produktu
- **Výrobcovia produktu**
  Testovacie scenáre: Nastavenie a overenie výrobcu produktu
- **Súvisiace produkty**
  Testovacie scenáre: Odstránenie všetkých súvisiacich produktov, Pridanie súvisiacich produktov, Overenie súvisiacich produktov
- **Varianty produktu**
  Testovacie scenáre: Vytvorenie variant produktu zo šablóny, Zrušenie používania variant produktu, Overenie variant produktu
- **Externé odkazy produktu**
  Testovacie scenáre: Odstránenie všetkých videí a odkazov produktu, Pridanie videí a odkazov produktu, Overenie videí a odkazov produktu

- **Príznaky produktu**
  Testovacie scenáre: Nastavenie príznakov produktu, Odstránenie všetkých príznakov produktu, Overenie príznakov produktu

B.4 Zľavy

Testovacie scenáre: Pridanie objemovej zľavy, Overenie validácie objemovej zľavy, Odstránenie všetkých objemových zliav, Test objemovej zľavy v košíku, Overenie cenníkov, Pridanie a overenie nového cenníka

B.5 Ostatné

Testovacie scenáre: Nastavenie štandardných nastavení v kategórii cenotvorba + DPH, Vyprázdnenie košíka po jednotlivých položkách